Algorithms for learning parsimonious context trees

Eggeling, Ralf

2019-06

Eggeling , R , Grosse , I & Koivisto , M 2019 , ' Algorithms for learning parsimonious context trees ' , Machine Learning , vol. 108 , no. 6 , pp. 879-911 . https://doi.org/10.1007/s10994-018-5770-9

http://hdl.handle.net/10138/302822
https://doi.org/10.1007/s10994-018-5770-9

Machine Learning (2019) 108:879–911
https://doi.org/10.1007/s10994-018-5770-9

Algorithms for learning parsimonious context trees

Ralf Eggeling[1,2] · **Ivo Grosse**[3] · **Mikko Koivisto**[1]

Received: 15 December 2017 / Accepted: 25 October 2018 / Published online: 12 November 2018

Abstract
Parsimonious context trees, PCTs, provide a sparse parameterization of conditional probability distributions. They are particularly powerful for modeling context-specific independencies in sequential discrete data. Learning PCTs from data is computationally hard due to the combinatorial explosion of the space of model structures as the number of predictor variables grows. Under the score-and-search paradigm, the fastest algorithm for finding an optimal PCT, prior to the present work, is based on dynamic programming. While the algorithm can handle small instances fast, it becomes infeasible already when there are half a dozen four-state predictor variables. Here, we show that common scoring functions enable the use of new algorithmic ideas, which can significantly expedite the dynamic programming algorithm on typical data. Specifically, we introduce a memoization technique, which exploits regularities within the predictor variables by equating different contexts associated with the same data subset, and a bound-and-prune technique, which exploits regularities within the response variable by pruning parts of the search space based on score upper bounds. On real-world data from recent applications of PCTs within computational biology the ideas are shown to reduce the traversed search space and the computation time by several orders of magnitude in typical cases.

Keywords Exact algorithms · Structure learning · Context-specific independence · Branch and bound · Sequential data

Editor: Jesse Davis.

Preliminary versions of this work have appeared in the proceedings of ICML 2015 (Eggeling et al. 2015a) and UAI 2016 (Eggeling and Koivisto 2016).

✉ Ralf Eggeling
eggeling@informatik.uni-tuebingen.de

[1] Department of Computer Science, University of Helsinki, Helsinki, Finland

[2] Present Address: Department of Computer Science, University of Tübingen, Tübingen, Germany

[3] Institute of Computer Science, Martin Luther University Halle–Wittenberg, Halle, Germany

1 Introduction

Univariate conditional distributions play a central role in various multivariate probabilistic models, such as Markov models, hidden Markov models, Bayesian networks, and general hierarchical graphical models. Ideally, each conditional distribution either involves only a few conditioning variables or one can assume the conditional distribution to take some simple form, for example, a linear model. In practice, neither case may apply, and we encounter the curse of dimensionality: the representation size of the conditional distribution, which usually is proportional to the number of free parameters, grows exponentially in the number of variables.

The concept of context-specific independence (Boutilier et al. 1996) provides an appealing approach to deal with the curse of dimensionality. Context-specific independence takes place when fixing some of the conditioning variables to certain states, called a *context*, the remaining variables provide no additional information about the response variable, that is, the response variable is independent of the rest given the context. Examples of general-purpose model classes that are based on the notion of context-specific independence include decision trees (Breiman et al. 1984; Quinlan 1986; Buntine 1992; Chipman et al. 1998), decision graphs (Oliver 1993; Chickering et al. 1997; Jaeger et al. 2006), chain event graphs (Smith and Anderson 2008), multi-linear functions (Chavira and Darwiche 2005), conditional independence trees (Su and Zhang 2005), and conditional probabilistic sentential decision diagrams (Shen et al. 2018).

When the explanatory variables are equipped with a natural linear ordering, more specialized models of context-specific independence are justified. *Context trees* (CTs) in particular enable computationally efficient learning of a sufficient set of contexts for categorical, linearly ordered random variables (Rissanen 1983; Volf and Willems 1994; Bühlmann and Wyner 1999) and have been applied in various scenarios that match this setting (Begleiter et al. 2004; Zhao et al. 2005; Ben-Gal et al. 2005). CTs arise from organizing the full conditional probability distribution as a rooted tree, where each node at layer ℓ corresponds to fixing the states of the first ℓ conditioning variables. Finding a CT that maximizes a given scoring function can be accomplished efficiently by pruning subtrees that are not justified by the observed data. While context trees excel in computational efficiency, their statistical efficiency decays when there are long-range dependencies and when the alphabet is non-binary.

To address the shortcoming of CTs, Bourguignon and Robelin (2004) proposed *parsimonious context trees* (PCTs). PCTs generalize CTs by identifying a context with a selection of state *subsets* for the explanatory variables, which yields a much wider range of admissible tree structures in relation to CTs (Fig. 1). This includes the important capability of (context-specifically) "skipping" a position partially or entirely, allowing in effect for a compact representation and statistically efficient learning even in the presence of long-range dependencies.

PCTs have found recent applications particularly within computational biology, where modeling sequential data over discrete alphabets constitutes a recurring challenge. Seifert et al. (2012) used PCTs for augmenting higher-order Hidden Markov models to improve Array-CGH analysis. Another well-studied application models DNA sequence patterns that are of importance for gene regulation (Eggeling et al. 2014a, 2015b). Here, PCTs augments an inhomogeneous Markov model that can be viewed a Bayesian network of fixed structure where the parents of each variable are the direct predecessors in the sequence.

Such an inhomogeneous parsimonious Markov model has several advantages for the given application domain. First, it yields favorable predictive performance in relation to alternative models such as Bayesian networks (Barash et al. 2003); see the study in Eggeling et al.

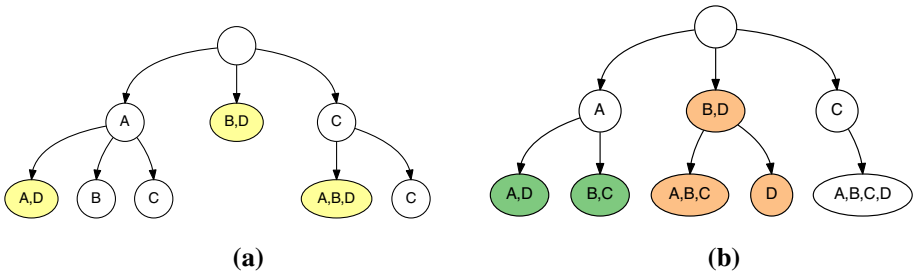

Fig. 1 Difference between CT and PCT. Consider two conditioning variables, both with alphabet $\Omega = \{A, B, C, D\}$. **a** A traditional CT. Nodes labeled with more than one symbol and highlighted in yellow combine events of sub-trees that have been pruned into so-called *pseudo nodes* (Bühlmann and Wyner 1999), ensuring that every possible event is still represented in the tree without changing the model itself. Due to the origin of the pseudo nodes there can be only one of them among a set siblings and subtrees below pseudo nodes are not allowed. **b** A PCT for the same problem size that relaxed both aforementioned restrictions. Here, more than one sibling node is allowed to be labeled by more than one symbol (green), and non-minimal subtrees are allowed below any node, irrespective of its label (orange) (Color figure online)

(2014a) and Sect. 7.8 in this article. Second, it can be used for unsupervised learning tasks, such as de novo motif discovery (Eggeling et al. 2014a, 2015b, 2017) or as component of a mixture model (Eggeling et al. 2017; Eggeling 2018), where learning is possible only through an iterative approach such as the EM algorithm (Dempster et al. 1977) or variants thereof (Nielsen 2000; Fujimaki and Morinaga 2012). Third, it allows for an intuitive model visualization through a conditional sequence logo (Eggeling et al. 2017) that is a direct generalization of the popular sequence logo (Schneider and Stephens 1990). Finally, it can be easily generalized to capture distal dependencies by relaxing the assumption of a fixed Bayesian network backbone structure (Eggeling 2018).

Irrespective of the concrete application, however, structure learning of PCTs is very challenging from a computational point of view, which has been considered a drawback of the model (Leonardi 2006). The reason for that are the relaxed structural constraints: even if a node is labeled by the full alphabet, which stands for context-specific independence, the node can be succeeded by a non-trivial subtree; hence the structure search cannot be stopped once some context-specific independencies have been found, but the whole space of possible structures has to be considered. Bourguignon and Robelin (2004) proposed a dynamic programming (DP) algorithm that is capable of finding a PCT of a given maximum depth d so as to maximize a given decomposable scoring function without explicitly enumerating all PCTs; a score is decomposable if it is the sum of so-called leaf scores. However, this algorithm still has to consider each potential leaf node that could occur in a valid PCT, the number of which grows exponentially in d.

In this article, we present techniques for enhancing the basic DP algorithm of Bourguignon and Robelin (2004), with the aim of significantly speeding up the structure learning of PCTs. Our central observation is that the basic DP algorithm makes essentially no assumptions about the structure of the scoring function. Put otherwise, for the common scoring functions used in practice, we should be able to enhance the algorithm by exploiting the particular form of the scoring function. Indeed, we will show that we can exploit regularities in the data to reduce the computational burden of finding an optimal PCT. There are two types of regularities, which can be capitalized upon by two different ideas respectively.

On the one hand, there are regularities among the realizations of the conditioning variables, which we can utilize: we store entire optimized subtrees—actually only their scores—in

memory for possible later re-use. This idea, we call *memoization*, has the drawback of being memory intensive. For that reason, we also investigate a parameterized extension of the idea that allows us to trade time for space.

On the other hand, there are regularities within the response variable. We exploit the regularities by devising two pruning rules, a *stopping rule* and a *deletion rule*, which allow us to ignore subproblems that are guaranteed to not contribute to an optimal PCT. The deletion rule resembles a simple pruning rule (Teyssier and Koller 2005) that is nowadays standard in structure learning in Bayesian networks: while that rule concerns the "is subset of" relation on candidate parent sets, our deletion rule concerns the "refines" relation on set partitions of the alphabet. To effectively apply the pruning rules in practice, we derive score upper bounds based on the properties of the considered concrete scoring functions, similar in spirit to the bounds of Tian (2000) and de Campos and Ji (2011) for structure learning in Bayesian networks.

We evaluate the performance of the individual techniques alone and in concert on real world data sets from the domain of computational biology. We use two data sets as running examples for demonstrating the detailed effects of parameter settings that control the algorithmic complexity. We further present an exhaustive study on a large variety of data sets that show a different degree of regularity among the input variables. These studies show that the proposed ideas can be highly effective in many cases, yielding speedup of up to two orders of magnitude for typical data sets.

This article is based on and considerably extends our preliminary work published in two conference papers (Eggeling et al. 2015a; Eggeling and Koivisto 2016). The first paper introduced the memoization idea, but it did not consider the parameterized extension that allows for trading time for space. The second paper introduced pruning techniques; however, the study was restricted to the BIC score (Schwarz 1978) and only derived a relatively simple bound that we will refer to as the *coarse* bound. The present work extends this path of research by deriving a substantially tighter bound, we will call the *fine* bound, and by making the bounds applicable also for other related scoring functions such as the AIC score (Akaike 1974). Due to these major methodological developments, the experimental studies are completely new, covering a larger number of data sets and instantiations of the proposed algorithms.

The remainder of this article is organized as follows. Section 2 contains a technical recap of PCTs, including a formal definition of the model and the structure learning problem, a description of the basic DP algorithm of Bourguignon and Robelin (2004), and a visual interpretation. In Sect. 3, we present the memoization technique in its plain variant as well as a parameterized version for limited-memory usage. We then describe the pruning ideas: Sect. 4 gives the upper bounds on the scoring function; the pruning rules that rely on these bounds are given in Sect. 5. Next, we describe the interplay of all different algorithmic ingredients in a final algorithm in Sect. 6. We report on the case studies in Sect. 7 and conclude the article with some discussions and final remarks in Sect. 8.

2 Parsimonious context trees

In this section, we revisit the definition of a parsimonious context tree (PCT) and a score-and-search approach to structure learning of PCTs. We also describe the basic dynamic programming algorithm of Bourguignon and Robelin (2004) and its interpretation as subtree selection in a so-called extended PCT.

2.1 Basic definitions

Let Ω be a finite set. A rooted, balanced, node-labeled tree of depth d is called a *parsimonious context tree (PCT) over* Ω if the node labels satisfy the following property: for each node at depth $\ell < d$ the labels of the node's children form a set partition of Ω, that is, the labels of the children are pairwise disjoint nonempty subsets of Ω whose union is Ω. We call the set Ω the *alphabet* and its members *symbols*.

We identify each node of a PCT with the sequence of labels $\mathbf{V} = V_\ell \cdots V_1$ of the nodes on the unique path from the node up to, but excluding, the root; here and henceforth we write the labels in the reversed order. We may interpret the node $\mathbf{V}$ as the set $\bigcup_{j\geq 0} \left(\Omega^j \times V_\ell \times \cdots \times V_1\right)$, which consists of the sequences over Ω whose length is at least ℓ and whose ith symbol belongs to V_i for $i = \ell, \ldots, 1$. Following this interpretation, we say that a sequence $\mathbf{x}$ *matches* $\mathbf{V}$ if $\mathbf{x} \in \mathbf{V}$. It follows that the leaves of a PCT of depth d correspond to a set partition of the set of all sequences over Ω whose length is at least d. Furthermore, each PCT corresponds to a distinct partition.

Given a PCT $\mathcal{T}$ and its node $\mathbf{V}$, we denote by $\mathcal{T}(\mathbf{V})$ the subtree of $\mathcal{T}$ rooted at $\mathbf{V}$. We say that the subtree is *minimal* if it consists of a single chain of nodes down to a single leaf, thus all nodes labeled by Ω; we say that the subtree is *maximal* if it consists only of nodes labeled by singletons $\{a\} \subseteq \Omega$, thus having $|\Omega|^{d-\ell(\mathbf{V})}$ leaves; here and henceforth $\ell(\mathbf{V})$ denotes the depth of node $\mathbf{V}$.

Now consider expressing the conditional distribution of a *response* variable y given a sequence of *explanatory* variables $\mathbf{x} = x_d \cdots x_1$, where for simplicity we assume all the variables take values from Ω. To specify a conditional distribution using a PCT $\mathcal{T}$, we equip each leaf $\mathbf{V}$ of $\mathcal{T}$ with $|\Omega|$ parameters $\theta_{\mathbf{V}a}$, one parameter for each $a \in \Omega$. We interpret $\theta_{\mathbf{V}a}$ as the probability that $y = a$ given that $\mathbf{x}$ matches $\mathbf{V}$. To model a *data set* $\mathbf{z} = (\mathbf{x}^t, y^t)_{t=1}^N$ we assume that, given $\mathbf{x}^t$, the response y^t is independent of the remainder of the data. Writing $\Theta_\mathcal{T}$ for the parameters and $leaves(\mathcal{T})$ for the set of leaves of $\mathcal{T}$, we obtain the likelihood function

$$L_\mathcal{T}(\Theta_\mathcal{T}) := \prod_{\mathbf{V}\in leaves(\mathcal{T})} \prod_{a\in\Omega} \theta_{\mathbf{V}a}^{N_{\mathbf{V}a}}, \tag{1}$$

where $N_{\mathbf{V}a}$ denotes the count of the response a in data points where the explanatory variables match $\mathbf{V}$:

$$N_{\mathbf{V}a} := |\{t : \mathbf{x}^t \in \mathbf{V} \text{ and } y^t = a\}|. \tag{2}$$

We will further denote $N_\mathbf{V} := \sum_{a\in\Omega} N_{\mathbf{V}a}$.

2.2 The structure learning problem

We consider a score-and-search approach to learning PCTs from given data. Suppose we are given a scoring function S that associates each PCT $\mathcal{T}$ of depth d with a real-valued score $S_\mathcal{T}$. An example of a practically relevant scoring function is the BIC score (Schwarz 1978), which takes the form of a penalized maximum-likelihood score:

$$S_\mathcal{T}^{\text{BIC}} = \max_{\Theta_\mathcal{T}} \left\{ \ln L_\mathcal{T}(\Theta_\mathcal{T}) \right\} - \frac{|leaves(\mathcal{T})|}{2} (|\Omega| - 1) \ln N.$$

Other scoring functions are, among others, the AIC score (Akaike 1974), the Bayes score with Dirichlet prior (Heckerman et al. 1995), and a factorized normalized maximum-likelihood

Table 1 Complexity of PCT learning

(a) Number of valid PCTs			*(b) Size of extended PCT*		
d	$\lvert\Omega\rvert = 3$	$\lvert\Omega\rvert = 4$	d	$\lvert\Omega\rvert = 3$	$\lvert\Omega\rvert = 4$
1	5	15	1	7	16
2	205	72,465	2	57	241
3	8.74×10^6	2.75×10^{19}	3	400	3616
4	6.68×10^{20}	5.78×10^{77}	4	2801	54,241
5	2.98×10^{62}	1.12×10^{311}	5	19,608	813,616

(fNML) score (Silander et al. 2010). Whatever the scoring function is, the task is to find an optimal PCT,

$$\mathcal{T}_* \in \operatorname*{argmax}_{\mathcal{T}} S_{\mathcal{T}}. \tag{3}$$

Aside from the fact that multiple PCTs may achieve the optimal score, this task is practically equivalent to the task of finding the optimal score $S_{\mathcal{T}_*}$. For convenience, we focus on the latter problem for the rest of the paper. We will see that an optimal PCT $\mathcal{T}_*$ can be constructed by standard routines with a small extra effort; we illustrate the basic idea in Sect. 2.4.

The main obstacle for solving the optimization problem of Eq. 3 in practice is the sheer number of possible PCTs, which grows very rapidly with depth and alphabet size. To be precise, for alphabet Ω and depth d, the number of valid PCTs, denoted by $T_\Omega(d)$, satisfies the recurrence

$$T_\Omega(d) = \sum_{k=1}^{|\Omega|} S_{|\Omega|,k} \cdot T_\Omega(d-1)^k, \tag{4}$$

where $S_{i,j}$ denotes the Stirling number of the second kind and $T_\Omega(0) = 1$. This recurrence holds because a depth-d PCT is obtained by joining some number $k \in \{1, \ldots, |\Omega|\}$ of arbitrary depth-$(d-1)$ PCTs and labeling their k roots by distinct subsets of Ω such that the labels form a set partition of Ω (whence the factor $S_{|\Omega|,k}$). Table 1a shows concrete values for $T_\Omega(d)$ for small $|\Omega|$ and d.

2.3 Basic dynamic programming

Bourguignon and Robelin (2004) presented a dynamic programming (DP) algorithm that finds the maximum score $S_{\mathcal{T}_*}$ without enumerating all distinct PCTs. The algorithm, we shall call *basic DP*, relies on the decomposability of the scoring function:

Assumption 1 *(Decomposability)* The scoring function S is given by

$$S_{\mathcal{T}} = \sum_{\mathbf{V} \in leaves(\mathcal{T})} S(\mathbf{V}),$$

where the leaf scores $S(\mathbf{V})$ depend on $\mathcal{T}$ only through their label sequence, and not on other structural properties of $\mathcal{T}$.

We formulate the property of decomposability as an assumption to emphasize its crucial role as an enabler of the algorithms we develop in the sequel. Nevertheless, decomposability is a mild assumption, satisfied by virtually all practically relevant scoring functions. For example, the BIC score S^{BIC} is decomposable with the leaf scores

$$S^{\text{BIC}}(\mathbf{V}) = \sum_{a \in \Omega} N_{\mathbf{V}a} \ln \frac{N_{\mathbf{V}a}}{N_{\mathbf{V}}} - \frac{1}{2}(|\Omega| - 1) \ln N.$$

It is worth noting that a decomposable scoring function is fully specified by the leaf scores, that is, a "local" scoring function that associates every possible leaf node with a real number (for a given data set). Accordingly, by a *leaf node*, without any reference to a particular PCT, we simply refer to a node that is a leaf in some PCT of a fixed depth. Similarly, an *inner node* will refer to a node that is an inner node in some PCT of a fixed depth.

To describe the algorithm, denote by $S_{\mathcal{T}(\mathbf{V})}$ the sum of the leaf scores in the subtree $\mathcal{T}(\mathbf{V})$ of $\mathcal{T}$ rooted at an inner node $\mathbf{V}$ of $\mathcal{T}$; for a leaf $\mathbf{V}$, put $S_{\mathcal{T}(\mathbf{V})} := S(\mathbf{V})$. We have the recurrence

$$S_{\mathcal{T}(\mathbf{V})} = \sum_{\text{child } \mathbf{C} \text{ of } \mathbf{V}} S_{\mathcal{T}(\mathbf{C})}, \tag{5}$$

and, in particular, $S_{\mathcal{T}} = S_{\mathcal{T}(\Lambda)}$, where Λ is the root node of $\mathcal{T}$. Exploiting this recurrence, the algorithm of Bourguignon and Robelin (2004) optimizes the score over the subtrees rooted at $\mathbf{C}$, independently for each possible child node $\mathbf{C}$, and then selects a set of children that form an optimal partition of the parent node $\mathbf{V}$.

More formally, for every node $\mathbf{V}$ we let $S_*(\mathbf{V})$ be the maximum score over PCT subtrees rooted at $\mathbf{V}$; in particular, for a leaf $\mathbf{V}$ we have $S_*(\mathbf{V}) = S(\mathbf{V})$, and at the root we have

$$S_*(\Lambda) = \max_{\mathcal{T}} S_{\mathcal{T}} = S_{\mathcal{T}_*}. \tag{6}$$

The next proposition establishes the recurrence used by the dynamic programming algorithm. It follows directly from the recurrence in Eq. 5.

Proposition 1 (Dynamic programming recurrence) *Let* $\mathbf{V}$ *be an inner node. Then*

$$S_*(\mathbf{V}) = \max_{\substack{\{C_1,\ldots,C_r\} \\ \textit{partition of } \Omega}} \left\{ S_*(C_1\mathbf{V}) + \cdots + S_*(C_r\mathbf{V}) \right\}. \tag{7}$$

2.4 Dynamic programming as search on extended PCT

The inner workings of the algorithm of Bourguignon and Robelin (2004) and the construction of the optimal PCT itself can be viewed as bottom-up reduction of a data structure called *extended PCT*, as illustrated in Fig. 2. In contrast to a PCT, the sibling nodes in an extended PCT do not partition their parent node, but are labeled by all nonempty subsets of Ω. An extended PCT thus contains all possible PCTs as subtrees.

The base case of the algorithm requires the computation of leaf scores of the extended PCT, and the task is then to reduce the extended PCT so that a maximum-score PCT remains. For each inner node it then (1) computes an optimal selection of children with the constraint that the node labels form a partition of Ω, and (2) removes all children not selected and the subtrees below. Since an inner node can be evaluated only once all of its children have already a score attached to them, the DP algorithm amounts to a bottom-up reduction of the extended PCT in a layer-wise fashion, as displayed in Fig. 2b. The algorithm terminates once an optimal selection of children of the root node are computed; a possible final result is shown in Fig. 2c.

The size of the extended PCT, which essentially determines the complexity of the DP algorithm, grows substantially slower than the number of possible PCTs (Table 1). Yet, the complexity of the algorithm is exponential in the maximum depth of the PCT, and over-exponential in the alphabet size: The algorithm computes the leaf scores of $(2^{|\Omega|} - 1)^d$

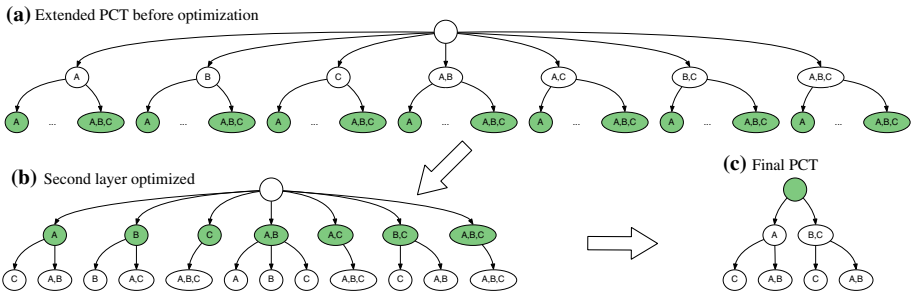

Fig. 2 Basic PCT optimization. We show the bottom-up reduction of the extended PCT for a toy example of $d = 2$ and $\Omega = \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$. **a** Initially only the leaf scores of the extended PCT have an exact, optimal score assigned to them. **b** For each set of sibling leaves, the optimal valid selection of children is computed, the non-contributing siblings are discarded, and the winning score is propagated upwards to become the score of the parent. **c** The same principle is applied on the higher layer in order to select the optimal children of the root, obtaining a valid PCT with optimal score

leaves of the extended PCT; in addition, it computes an optimal selection of children for $\sum_{\ell=0}^{d-1}(2^{|\Omega|}-1)^{\ell}$ inner nodes, each of which takes $O(3^{|\Omega|})$ time using a routine we describe in Sect. 5.2.

2.5 Learning with weighted data

Suppose each data point $(\mathbf{x}^t, y^t)$, with $t = 1, \ldots, N$, is associated with a real-valued weight $w^t \geq 0$. The weights may arise from different origins: First, scientific experiments, such as modern high-throughput technologies in DNA sequence analysis (Orenstein and Shamir 2014), may directly produce weighted data. Second, it can be more efficient to store an original data set with many duplicates as weighted data consisting of unique data points where the weight equals the number of occurrences in the original data set. Third, learning with weighted data is needed when the model is a component of a mixture model that is learned using the EM algorithm or variants thereof (Fujimaki and Morinaga 2012); see Eggeling (2018) for a recent application of PCTs in such a scenario.

To make the methods presented in this and later sections applicable to weighted data, it suffices to generalize the definition of integer counts in Eq. 2 to weighted counts:

$$N_{\mathbf{V}a} := \sum_{t:\mathbf{x}^t \in \mathbf{V} \text{ and } y^t = a} w^t. \tag{8}$$

We obtain Eq. 2 as a special case when all weights are equal to 1. It is worth noting that for weighted data the sample size is defined as the total weight $\sum_{t=1}^{N} w^t$ instead of the number of (weighted) data points. Thus, for example, in the penalty term of the BIC score, N is replaced by the total weight.

3 Memoization

The basic dynamic programming algorithm of Bourguignon and Robelin (2004), described in the previous section, only exploits the decomposability of the scoring function (Assumption 1). Fortunately, the commonly used scoring functions also share other properties that

enable further computational savings. In this section, we formalize a sufficient condition under which two different subproblems (i.e. nodes of the extended PCT) must have equal solutions and thus need to be solved only once; we call the resulting technique *memoization*.

3.1 Storing and reusing solved subproblems

For a node $\mathbf{V}$ of an extended PCT, write $I(\mathbf{V}) := \{t : \mathbf{x}^t \in \mathbf{V}\}$ for the set of indices of data points that match the node. We will make use of the following *data locality* property, which we, again, formalize as an assumption.

Assumption 2 *(Data locality)* If two leaves $\mathbf{V}$ and $\mathbf{V}'$ of an extended PCT are matched by the same data points, $I(\mathbf{V}) = I(\mathbf{V}')$, then their scores are equal, $S(\mathbf{V}) = S(\mathbf{V}')$.

In essence, data locality means that the leaf score $S(\mathbf{V})$ depends on the leaf $\mathbf{V}$ only through the data subset indicated by $I(\mathbf{V})$. If the property holds for the leaf scores, then, due to Eq. 7, it also holds for the optimal scores of the inner nodes of any fixed depth:

Proposition 2 (Memoization) *Let* $\mathbf{V}$ *and* $\mathbf{V}'$ *be two nodes such that* $I(\mathbf{V}) = I(\mathbf{V}')$ *and* $\ell(\mathbf{V}) = \ell(\mathbf{V}')$. *Then* $S_*(\mathbf{V}) = S_*(\mathbf{V}')$.

Assumption 2 is fulfilled, for instance, by the BIC score, and more generally by most practically relevant scoring functions that can be written in terms of a penalized maximum-likelihood score. A notable exception, however, is the Bayes score with context-dependent hyperparameters (Eggeling et al. 2014b), comparable to the family of BDeu scores for Bayesian networks (Heckerman et al. 1995).

Proposition 2 enables the following rule for speeding up the dynamic programming algorithm. Consider an inner node $\mathbf{V}$ of the extended PCT. Suppose there exists another node $\mathbf{V}'$ at the same depth $\ell(\mathbf{V}') = \ell(\mathbf{V})$ and the same data subset index $I(\mathbf{V}') = I(\mathbf{V})$. Then the optimal scores of the two nodes are equal due to Proposition 2. Thus it suffices to compute the optimal score only once for the node that is visited first, say $\mathbf{V}$, and store it in an appropriate data structure. To this end, we use a hash map, with the pair $(I(\mathbf{V}), \ell(\mathbf{V}))$ as the key and the score $S_*(\mathbf{V})$ as the value. For node $\mathbf{V}'$, repeated computations are avoided by calling the value from the data structure by the key $(I(\mathbf{V}'), \ell(\mathbf{V}'))$. Figure 3 shows an example where the absence of one particular pattern in the data leads to three applications of the rule so that only 4 of 7 subtrees of a common parent node need to be explicitly computed. Note that, in general, the rule is not limited to sibling nodes, but may well apply among more distantly related nodes.

The effectiveness of the memoization rule is data dependent. For small data sets but deep trees, the rule is expected to apply often, for then the number of nodes gets large while the number of distinct data subsets that match high-depth nodes gets small. Likewise, the relative gain is expected be the higher, the larger the alphabet is. Also, the memoization rule is likely to apply more frequently on highly structured data than on random data.

3.2 Trading time for space

A downside of the memoization rule is an increased memory consumption due to the necessity to store the computed optimal scores of the visited nodes in the extended PCT in memory for potential future reuse. In order to find an appropriate tradeoff between memory and time consumption, it is reasonable to control the degree of memoization employed by the algorithm.

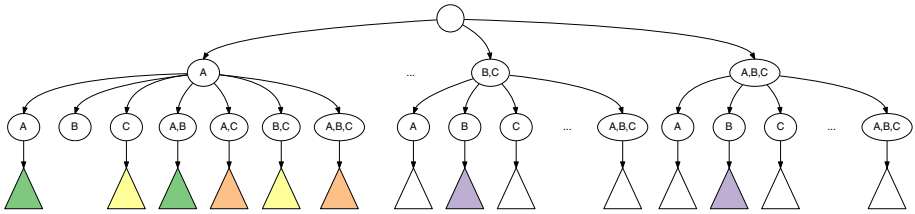

Fig. 3 Example of memoization. Consider the shown part of the extended PCT and assume that no data word ends with BA. First, we do not need to compute the subtree below the second layer node corresponding to suffix BA. More important, the first node at depth two within the first leftmost subtree represents the same data subset as its sibling node with label {A, B} and so the subtrees below both nodes are identical (green). The same applies for two further pairs of siblings within the same subtree (yellow and orange). The two subtrees rooted by {B,C} (middle) and {A,B,C} (right) show an example where the same missing data word causes data subsets among "cousins", both labeled by {B}, to be identical. The same situation applies in principle also to subtrees below depth-one-nodes {C} and {A,C}, as well as, {B} and {B,C} (not shown due to space limitations) (Color figure online)

The key idea is to store the optimal score of node $\mathbf{V}$ only if it is likely to be re-used and holds a promise for significant savings in running time. While there are several possibilities to make such a decision, for instance, based on the number of (distinct) data points matching $\mathbf{V}$, we have found that the depth of $\mathbf{V}$ is the decisive factor: there are only a few shallow nodes in the extended PCT and the potential savings in running time are immense when reusing applies, since a shallow node is root of a large subtree that needs to be traversed. On the other hand, there is a vast number of deep nodes, for which the potential savings are comparatively small as they are parents of only very small subtrees. Hence, it is reasonable to limit the maximum depth at which scores are stored in memory by an external *memoization depth* parameter, denoted by m, which we can use to trade time against space. We empirically investigate the effect of varying m in Sect. 7.5.

4 Score upper bounds

In this section, we present techniques to prune parts of the search space based on fast-to-compute upper bounds on the optimal scores of subproblems (i.e. nodes of an extended PCT). To this end, we will make yet another assumption regarding the scoring function, in addition to Assumptions 1 and 2.

Assumption 3 *(Constant leaf penalty)* The leaf score takes the form of a penalized maximum log-likelihood,

$$S(\mathbf{V}) \;=\; L(\mathbf{V}) - K,$$

where $L(\mathbf{V}) = \sum_{a\in\Omega} N_{\mathbf{V}a} \ln \frac{N_{\mathbf{V}a}}{N_{\mathbf{V}}}$ is the maximum log-likelihood of leaf $\mathbf{V}$ and K is a constant independent of $\mathbf{V}$. The function $L(\mathbf{V})$ naturally extends also to every inner node $\mathbf{V}$.

Note that the constant K is allowed to depend on the data size and the size of the alphabet—we only require that K is the same number for different choices of $\mathbf{V}$. The assumption of constant leaf penalty is fulfilled, for instance, by the BIC score, with $K = \frac{1}{2}\big(|\Omega| - 1\big) \ln N$, and by the AIC score, with $K = \big(|\Omega| - 1\big)$. An example of a scoring function that fulfills Assumptions 1 and 2 but not Assumption 3 is the fNML score (Silander et al. 2010): while the fNML score takes the form of a penalized maximum log-likelihood, the penalty term is the multinomial regret function and thus depends on the count $N_{\mathbf{V}}$.

For the remainder of the paper, we assume Assumption 3 to be fulfilled implicitly for every score S mentioned, which can thus be BIC, AIC, or any other scoring criteria arising from different choices for the constant leaf penalty K.

4.1 A simple bound

First, we introduce a simple, coarse upper bound that requires no substantial pre-computations and can thus always be used at no additional costs. Consider an inner node $\mathbf{V}$. To upper bound the maximum score over all possible subtrees rooted at $\mathbf{V}$, we upper bound the largest possible gain in the maximum-likelihood term on one hand, and lower bound the inevitable penalty due to increased model complexity on the other hand.

Consider first the likelihood term. We make use of the observation that every PCT is *nested* in the maximal PCT, which has $|\Omega|^d$ leaves, each matching a single realization $\mathbf{x} \in \Omega^d$ of the explanatory variables. The same holds also locally for any PCT subtree below the node $\mathbf{V}$. Hence, the likelihood term is maximized by the maximal model, which partitions the set $\Omega^{d-\ell(\mathbf{V})}$ into singletons. We obtain the upper bound

$$L^{\text{UB}}(\mathbf{V}) := \sum_{\mathbf{a} \in \Omega^{d-\ell(\mathbf{V})}} L(\mathbf{aV}), \tag{9}$$

that is, the maximum likelihood obtained by any PCT subtree rooted at $\mathbf{V}$ is at most $L^{\text{UB}}(\mathbf{V})$. Here, $\mathbf{aV}$ denotes the leaf node obtained by extending the context of node $\mathbf{V}$ with a single realization $\mathbf{a}$ of the remaining explanatory variables. Note that $L^{\text{UB}}(\mathbf{V})$ can be computed fast by only summing over the sequences $\mathbf{a}$ that occur in the data points that match $\mathbf{V}$.

Consider then the penalty term. We distinguish two cases:

Case 1 The minimal subtree is optimal. In this case, we can compute the exact score directly: $S_*(\mathbf{V}) = L(\mathbf{V}) - K$.

Case 2 The minimal subtree is not optimal. Thus an optimal subtree makes at least one split, and therefore has at least two leaves. In this case, the likelihood term is upper bounded by $L^{\text{UB}}(\mathbf{V})$, while the penalty term is at least $2K$.

Combining the two cases yields the following bound.

Proposition 3 (Coarse upper bound) *Let* $\mathbf{V}$ *be an inner node. Denote*

$$S_{\text{coarse}}(\mathbf{V}) := \max\{L(\mathbf{V}) - K, L^{UB}(\mathbf{V}) - 2K\}.$$

Then

$$S_*(\mathbf{V}) \leq S_{\text{coarse}}(\mathbf{V}).$$

While this upper bound is coarse and sometimes tight, we next consider two possibilities to further tighten it significantly by investing more effort in pre-computations.

4.2 Refining the bound

Our finer upper bound stems from a simple observation:

A PCT with n leaves can split according to at most $n - 1$ *explanatory variables.*

For example, if a PCT has 2 leaves (implying a penalty term $2K$), then the likelihood upper bound of Eq. 9 allowing splits according to all explanatory variables can be overly loose,

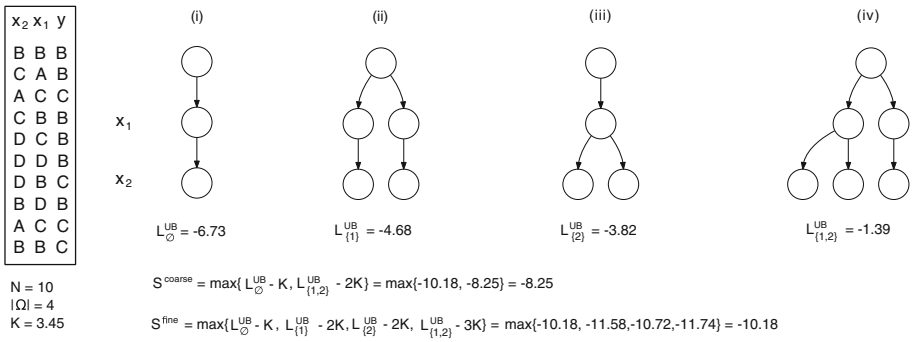

Fig. 4 Upper bounds for a small example data set (box)

for we actually can split according to at most one variable. To formalize this idea, for every $\ell = 1, 2, \ldots, d$ and index subset $J \subseteq \{1, 2, \ldots, d - \ell\}$ let $\mathcal{A}(\ell, J)$ denote the family of all subsets of sequences $\mathbf{A} \subseteq \Omega^{d-\ell}$ such that A_i is a singleton if $i \in J$ and $A_i = \Omega$ otherwise. In other words, the family forms a set partition of $\Omega^{d-\ell}$ into $|\Omega|^{|J|}$ disjoint sets according to the explanatory variables indexed in J (shifted by ℓ). For any inner node $\mathbf{V}$ we obtain the upper bounds

$$L_J^{\text{UB}}(\mathbf{V}) := \sum_{\mathbf{A} \in \mathcal{A}(\ell(\mathbf{V}), J)} L(\mathbf{AV}) \tag{10}$$

that is, the maximum likelihood obtained by any PCT subtree rooted at $\mathbf{V}$ is at most $L_J^{\text{UB}}(\mathbf{V})$, provided that the subtree splits only according to variables indexed by J. As a special case of these bounds, we obtain the coarse upper bound by setting $J = \{1, 2, \ldots, d - \ell(\mathbf{V})\}$.

By maximizing the bound over the sets J we obtain the following bound for the score:

Proposition 4 (Fine upper bound) *Let* $\mathbf{V}$ *be an inner node. Denote*

$$S_{\text{fine}}(\mathbf{V}) := \max_{J \subseteq \{1,2,\ldots,d-\ell(\mathbf{V})\}} L_J^{UB}(\mathbf{V}) - (|J| + 1)K. \tag{11}$$

Then

$$S_*(\mathbf{V}) \leq S_{\text{fine}}(\mathbf{V}) \leq S_{\text{coarse}}(\mathbf{V}).$$

The number of likelihood-terms that have to be computed is thus $2^{d-\ell(\mathbf{V})}$ instead of two for the coarse upper bound, which may appear as a substantial additional investment. However, the greatest additional computational effort, 2^d likelihood computations, occurs solely at the root of the extended PCT, whereas closer to the leaves the computational effort decreases very rapidly as $\ell(\mathbf{V})$ increases. Since the number of nodes in the extended PCT grows faster than 2^d, we may expect the amortized additional effort due to the fine upper bound to amount only to a small overhead—a low price for potentially much tighter bounds. We empirically study the practical effect of the fine bound on running times in Sect. 7.7.

We show an example that compares the fine and coarse upper bound for a small data set of $N = 10$ over the alphabet $\Omega = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}\}$ in Fig. 4. Using the BIC score, the value of the constant leaf penalty is $K = \frac{3}{2} \ln 10 \simeq 3.45$. We consider bounding the score of the root node and thus omit explicit reference to the argument $\mathbf{V}$ of the upper bound. Since there are two explanatory variables x_1 and x_2, we distinguish four cases, namely (1) full independence of y from the explanatory variables, (2) independence from x_2 (but not x_1), (3) independence

from x_1 (but not x_2), (4) dependence on both variables. For each of these four cases, we show the smallest possible PCT topology and the maximal possible likelihood for the data set. Note that each L^{UB} stems from splitting the data into a larger number of groups than the number of leaves in the smallest possible PCT of the same case; however, for both the splits concern exactly the same explanatory variables. We find that in this example the coarse upper bound is too optimistic: while there is a certain dependence in the data, it requires both explanatory variables to fully utilize it. The fine bound shows that doing so does not give a better score than the independence model, knowledge that can be utilized for pruning the search space (Sect. 5).

4.3 Lookahead

The upper bounds, both the coarse and the fine bound, can be computed directly for a given node without entering the recursion. However, we can further tighten the bounds, if we do enter the recursion for one or more steps, in effect, performing a lookahead on the data. To this end, for every node $\mathbf{V}$ and number of steps $q = 1, 2, \ldots, d - \ell(\mathbf{V})$, define the *q-step lookahead upper bound* as

$$S_q(\mathbf{V}) := \max_{\substack{\{C_1,\ldots,C_r\} \\ \text{partition of } \Omega}} \sum_{i=1}^{r} S_{q-1}(C_i\mathbf{V}), \tag{12}$$

with S_0 being the base case of a flat upper bound, say S_{coarse} or S_{fine}.

Proposition 5 (Lookahead bound) *Let* $\mathbf{V}$ *be an inner node and* $q \in \{1, 2, \ldots, d - \ell(\mathbf{V})\}$. *Then*

$$S_*(\mathbf{V}) \leq S_q(\mathbf{V}) \leq S_0(\mathbf{V}). \tag{13}$$

Using the lookahead bound with a large q constitutes a substantial computational effort. Specifically, if $q = d - \ell(\mathbf{V})$, the bound equals the optimal score and is, in essence, obtained by traversing through all possible PCT subtrees. Hence, the choice of q could be critical for obtaining a good trade-off between gained savings and additional invested effort in relation to the flat bound. We will investigate this issue empirically in Sect. 7.3.

5 Pruning rules

Armed with the score upper bounds derived in the previous section, we next present pruning rules that aim at deciding at each visited node in the extended PCT, whether the upper bounds allow us to avoid exact solving of the corresponding subproblem.

5.1 Stopping rule

We begin with a simple pruning rule, which follows directly from the aforementioned upper bound. The idea can be phrased as follows:

> Stop the search at a node when context-specific independence can be declared already without explicitly considering the possible subtrees.

From this basic idea it follows that using a lookahead bound within the stopping rule is pointless. While it may yield a tighter bound, it has to consider subtrees explicitly and solve the alphabet partition problem at least once, which are the tasks that are to be avoided. Formally, we have the following.

Proposition 6 (Stopping rule) *Let* $\mathbf{V}$ *be an inner node, and let* $\mathcal{T}'$ *be the minimal subtree rooted at* $\mathbf{V}$. *If*

$$S_0(\mathbf{V}) = L(\mathbf{V}) - K,$$

then $S_*(\mathbf{V}) = S_{\mathcal{T}'}(\mathbf{V})$.

The correctness can be shown by contradiction. Assume $\mathcal{T}'$ does not yield the optimal score. Then $L(\mathbf{V}) - K < S_*(\mathbf{V})$. But since $S_*(\mathbf{V}) \leq S_0(\mathbf{V})$, this violates the premise.

5.2 Deletion rule

The idea of our second rule is to identify a node with such a low score that the node cannot appear in any optimal PCT. In an idealized form, the rule reads as follows:

> Delete a child node if the best set of children it belongs to is worse than some other set of children (to which the node does not belong).

As we wish to delete as many potential child nodes as possible and not compute their optimal scores, we cannot assume the optimal scores of the sibling nodes are available. Thus, to make the rule concrete, we resort to upper bounds on the scores. Likewise, we need to lower bound the optimal score among the valid sets of children. While, in principle, various lower bounding schemes would be possible, we have chosen to use a particularly simple lower bound: the optimal score of the Ω-labeled child. We next describe the bounds and the rule more formally.

Consider a node $\mathbf{V}$. To efficiently check whether a child node $C\mathbf{V}$ can be deleted, we need an upper bound on the score obtained by a partition of $\mathbf{V}$ that includes $C\mathbf{V}$. To this end, let us associate any set function $f : 2^{\Omega} \to \mathbb{R}$ with another function $f^* : 2^{\Omega} \to \mathbb{R}$ defined by letting $f^*(\emptyset) := 0$ and, for all $\emptyset \subset B \subseteq \Omega$,

$$f^*(B) := \max_{\substack{\{C_1,\ldots,C_r\} \\ \text{partition of } B}} \big\{ f(C_1) + \cdots + f(C_r) \big\}. \tag{14}$$

We will set each value $f(C)$ to the score upper bound of $C\mathbf{V}$ at a fixed node $\mathbf{V}$. Observe that in the important special case when the values $f(C)$ are the optimal scores, the value $f^*(\Omega)$ equals the optimal score of node $\mathbf{V}$, following Eq. 7.

Computing f^* for a given f may be slow if done in a brute-force fashion, for the number of set partitions grows super-exponentially in the size of the ground set Ω. Fortunately, there is a faster, dynamic programming algorithm that runs in $O(3^{|\Omega|})$ time. The algorithm is based on the recurrence

$$f^*(B) = \max_{\emptyset \subset C \subseteq B} \big\{ f(C) + f^*(B \backslash C) \big\}. \tag{15}$$

This recurrence holds because an optimal set partition $\{C_1, \ldots, C_r\}$ of B must include a set $C_1 \subseteq B$ such that $\{C_2, \ldots, C_r\}$ is an optimal set partition of $B \backslash C_1$. The algorithm requires a constant time for each of the $3^{|\Omega|}$ pairs (B, C) satisfying $C \subseteq B$. This enables implementing the following deletion rule in $O(3^{|\Omega|})$ time.

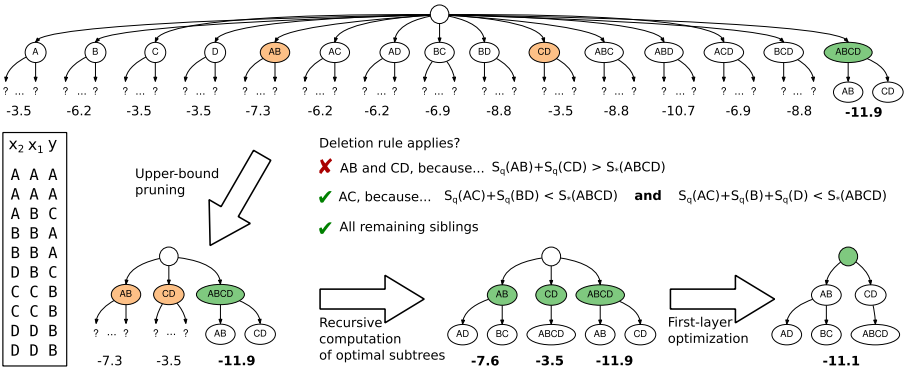

Fig. 5 Example of the deletion rule using BIC scores. We consider a small data set of $N = 10$ for two explanatory variables over the alphabet $\Omega = \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}\}$ (bottom-left box). The root of each subtree for which exact scores have been computed already is marked in green and corresponding score displayed in boldface below the subtree. The remaining other subtrees are associated with an upper bound on the score. The root of a subtree which contributes to an upper bounded partition score that is greater than the exact score of the maximal sibling node, and thus has to optimized explicitly, is displayed in orange (Color figure online)

Proposition 7 (Deletion rule) *Let* $\mathbf{V}$ *be an inner node, and let* $\emptyset \subset C \subset \Omega$. *Let* $f(C') := S_q(C'\mathbf{V})$ *for all* $\emptyset \subset C' \subseteq \Omega$. *If*

$$S_q(C\mathbf{V}) + f^*(\Omega \backslash C) \; < \; S_*(\Omega\mathbf{V}),$$

then the node $C\mathbf{V}$ *does not belong to any optimal PCT.*

The correctness can be shown by contradiction. Suppose $C\mathbf{V}$ did belong to an optimal PCT even though the premise was fulfilled. Then $S_*(C\mathbf{V}) + f^*(\Omega\backslash C) \geq S_*(\Omega\mathbf{V})$, leading to $S_q(C\mathbf{V}) < S_*(C\mathbf{V})$, which violates the property of S_q being an upper bound of S_*.

We illustrate the deletion rule by a small toy example in Fig. 5. Here, we focus on the first layer, where only for the maximal node an optimal PCT subtree is computed and thus an exact score is available already, whereas for the other siblings only upper bounded scores exist. Based on those scores, we compute the upper bounded scores of every possible partition and observe that only one partition, consisting of the two child nodes AB and CD, has an upper bounded score that is greater than the exact score of the maximal sibling node ABCD. All other siblings cannot contribute to an optimal partition of child nodes, as even the best partition they contribute to has an upper bounded score smaller than what is already achieved. Hence, the corresponding subtrees do not need to be optimized explicitly and can be deleted.

The deletion rule invests a certain amount of effort for which the obtained savings need to compensate before the rule becomes effective: In the worst case, we need to compute the optimal partition of children twice for each inner node, once with the upper bounded scores for excluding subtrees from further optimization, and once with the exact scores. As a positive side note, we observe that, while we focus on upper bounds based on Assumption 3 in this work, the deletion rule is in principle independent of the used scoring function, as long as an effective upper bound $S_q \leq S_*$ can be specified.

6 The final algorithm

We combine the presented ideas using pseudo code. Consider first the task of computing, for all nonempty subsets $B \subseteq \Omega$, the maximum total score over all partitions of B, in other words, the set function f^* for a given set function f, as defined in Eq. 14. The procedure MAX- PARTITION given below completes this task based on the recurrence in Eq. 15.

```
MAX- PARTITION(f)
1  g[∅] ← 0
2  for each ∅ ⊂ B ⊆ Ω in quasi-lexicographical order
3        do g[B] ← −∞
4           for each ∅ ⊂ C ⊆ B
5                 do g[B] ← max{g[B], f[C]+g[B\C]}
6  return g
```

The main algorithm, given below as procedure MAX- PCT, calls MAX- PARTITION(f) both with exact scores and with upper bounds, as specified by the argument f. The call MAX- PCT($\mathbf{V}$) returns the optimal score $S_*(\mathbf{V})$. We thus obtain the maximum score over all PCTs of depth d by calling MAX- PCT(Λ). Note that the pseudo-code assumes the concrete scoring function, defined by the constant leaf penalty K, to be set by the user. The same holds for the flat upper bound S_0, which may be coarse or fine, and which also serves as base case for lookahead upper bound S_q.

```
MAX- PCT(V)
 1  score ← L(V) − K
 2  if ℓ(V) < d and score < S_0(V) // false if stopping rule applies
 3     then s[Ω] ← MAX- PCT(ΩV)
 4          for each ∅ ⊂ B ⊆ Ω
 5                do u[C] ← S_q(CV)
 6          u′ ← MAX- PARTITION(u)
 7          for each ∅ ⊂ B ⊆ Ω
 8                do s[C] ← −∞
 9                   if u[C] + u′[Ω\C] > s[Ω] // false if deletion rule applies
10                      then s[C] ← MAX- PCT(CV)
11          s′ ← MAX- PARTITION(s)
12          score ← s′[Ω]
13 return score
```

While also omitted in the pseudocode for brevity, incorporating the memoization into the proposed algorithm is straightforward. We can add a test that checks whether the index set $I(\mathbf{V})$ has already occurred with some other node at the same depth directly when entering the function MAX- PCT($\mathbf{V}$). If the test is positive, the score is re-used and the rest of the function is skipped. If the test is negative and the depth of $\mathbf{V}$ is not larger than m, the score is stored in a hash data structure at the end of the function with the current data subset (index set) and the depth of $\mathbf{V}$ as the key.

7 Case studies

In the empirical part of this work, we evaluate the effects of the proposed techniques for expediting PCT learning using a Java-implementation based on the Jstacs library (Grau et al. 2012). The software is available at http://www.jstacs.de/index.php/PCTLearn.

7.1 Data

We consider the problem of modeling DNA binding sites of regulatory proteins such as transcription factors, which constitutes one established application of PCTs. A data set of DNA binding sites consists of short sequences of same length over the alphabet $\Omega = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ that are considered to be recognized by the same DNA-binding protein. In this application, the task is to model the conditional probability of observing a particular symbol at a certain position in the sequence given its direct predecessors—a task that directly fits to the setting outlined in Sect. 1. The probability of the full sequence is, by the chain rule, simply the product over all conditionals. Due to the nature of protein–DNA interaction, the conditional distribution at a particular position is strictly position-specific, so we need to learn a separate PCT for every sequence position in a data set.

We use data from the publicly available data base JASPAR (Sandelin et al. 2004), which contains a large number of DNA binding site data sets for various organisms. For the majority of this section, we focus on two exemplary data sets, which contain binding sites of human DNA-binding proteins called *CTCF* and *REST*. The sequence in both data sets are rather long (19 and 21 nucleotides), so there are quite a few PCTs of large depth to be learned. For conveniently referring to a particular learned PCT, we introduce the abbreviations CTCF-j for the PCT learned at the jth position of the CTCF data set, and REST-j likewise. Both proteins are known to recognize a rather complex sequence pattern (Eggeling et al. 2015b), which makes the structure learning problem challenging.

Figure 6 displays the position-specific marginal frequencies of both exemplary data sets in sequence logo representation of Schneider and Stephens (1990). They slightly differ in the length of the sequence, otherwise the properties are rather similar: both contain several highly informative positions, where the marginal distribution clearly favors a single symbol. But there is also an at least equally large number of positions where the marginal distribution contains only little information. The biggest difference among both data sets is the sample size N, that is, the number of sequences available to estimate the distributions from: for CTCF we have $N = 908$, for REST we have $N = 1575$.

For both data sets, we now learn optimal PCTs according to the BIC score setting the maximum depth to $d = 6$, except for the first six sequence positions, where the maximum

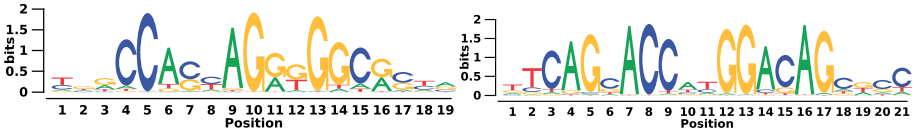

Fig. 6 Sequence logos of exemplary data sets that show position-specific marginals of relative frequencies of observations in both data sets. The height of the symbols in each stack are scaled relative to each other according to the relative frequencies; the total height of each stack is scaled inversely proportional to the entropy of the marginal (Schneider and Stephens 1990). **a** CTCF, **b** REST (Color figure online)

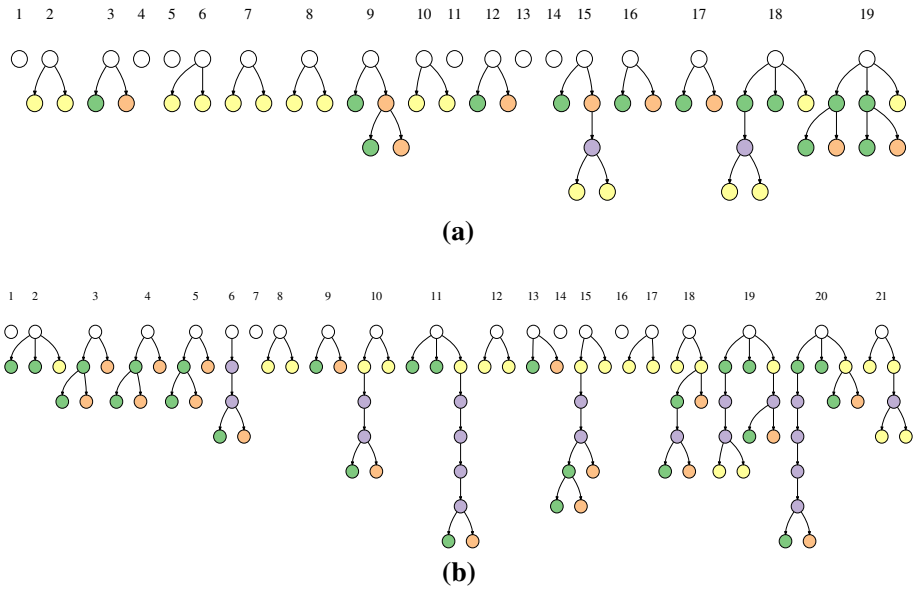

Fig. 7 Topologies of learned PCTs for exemplary data sets CTCF (top) and REST (bottom). The ith tree refers to the ith random variable in the sequence and the context variables are the direct predecessors. They are ordered so that a descendent of the root at depth j refers to state of the variable at sequence position $i - j$. The color of a node corresponds to the number of symbols in the node's label: green indicates a singleton, yellow corresponds to two, and orange to three symbols. Purple indicates the full alphabet; the node is only shown if it is a root of a non-minimal subtree. **a** CTCF, **b** REST (Color figure online)

depth is limited by the number of available explanatory variables. We show the resulting PCT structures in Fig. 7, hereby omitting the node labels in order to obtain a compact representation. Each node is still colored according to the size of its label, that is, whether it represents a singleton, the full alphabet, or a case in between.

We observe that the complexities of the optimal PCTs differ. In both data sets, there are sequence positions where a PCT that represents full statistical independence of the variable giving its predecessors is optimal according to the BIC score, which typically, though not always, occurs at highly informative positions. For CTCF all optimal PCTs have splits until at most depth three, whereas in the case of REST the allowed maximum depth of 6 is actually used to full capacity in case of REST-11 and REST-20, one final split occurs at depth 5, and three final splits at depth 4. The preference of REST for deeper trees, in comparison to CTCF, may be caused by a combination a larger sample size, which allows a bit higher model complexity, and the location of the highly informative positions in clusters, which spatially separates low-informative positions among whose dependencies are likely to occur.

The height and shape of the optimal PCT structures suggest that the PCT optimization for REST is generally computationally harder than for CTCF. In the following sections, we utilize both data sets for evaluating the effectiveness of the proposed memoization and pruning techniques.

7.2 Pruning versus memoization

In a first study, we compare the effect of memoization in its maximal variant, pruning with the fine upper bound and one-step lookahead ($q = 1$), and the combination of both techniques

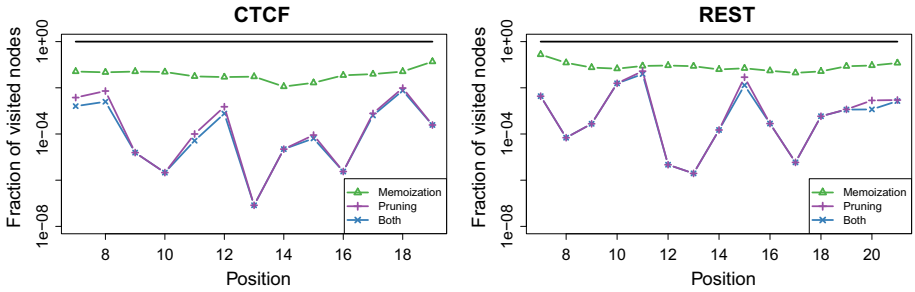

Fig. 8 Comparison of pruning and memoization. Shown is the fraction of visited nodes of extended PCT for different algorithm variants. The base case is the basic DP algorithm, which traverses the entire extended PCT (about 1.22×10^7 nodes) (Color figure online)

for finding optimal PCTs of maximum depth $d = 6$. For each position $j > 6$ in both data sets, we count the number of *visited nodes*, which are nodes in the extended PCT that are explicitly created (including lookahead nodes), and plot the savings achieved by each algorithmic variant in relation to the basic DP algorithm of Sect. 2.3 in Fig. 8. We observe that the general pattern is similar for both data sets.

Memoization reduces the search space by approximately one order of magnitude on average, and the savings vary only little from position to position. This can be explained by the structure of the data sets, where most positions have both high- and low-informative positions as predecessors, so the potential for exploiting regularities in the explanatory variables is in a similar range.

The effect of pruning, however, varies to a large degree. As a rule-of-thumb, at high-information positions pruning yields a tremendous reduction of the search space. In one exceptional case, CTCF-13, it is possible to prune already at the root, which we cannot always expect to happen: other positions with minimal optimal tree displayed in Fig. 7(top) require more effort to declare statistical independence. The savings at low-information positions are not as pronounced, but for all 28 cases under consideration, pruning yields higher savings than memoization.

It is thus no surprise that the combination of both is dominated by the effect of pruning: Memoization contributes only small additional savings for positions where pruning is not overly effective, such as CTCF-8 or REST-15.

Comparing the two data sets to each other, we find that the aggregated savings for CTCF are higher than for REST, which confirms the speculation from the previous section. In particular, for REST-11 and REST-15 finding optimal PCTs is relatively demanding. However, the optimal tree structure only implies a tendency, the correlation is not perfect: REST-7 and REST-20 seem equally challenging instances, yet the former yields a minimal optimal tree, whereas the latter yields an optimal tree with five leaves that reaches up to depth 6.

7.3 Pruning variants in detail

In the last section, we have seen that pruning with the fine upper bound and one-step lookahead is very competitive and that adding memoization on top of that yields only marginal additional savings. Now, we take a closer look at pruning itself in order to evaluate how large the impact of the different variants is. We compare the cross-combinations of (1) the coarse and fine upper bound and (2) q-step lookahead with $q \in \{0, 1, 2\}$. The results are shown in Fig. 9.

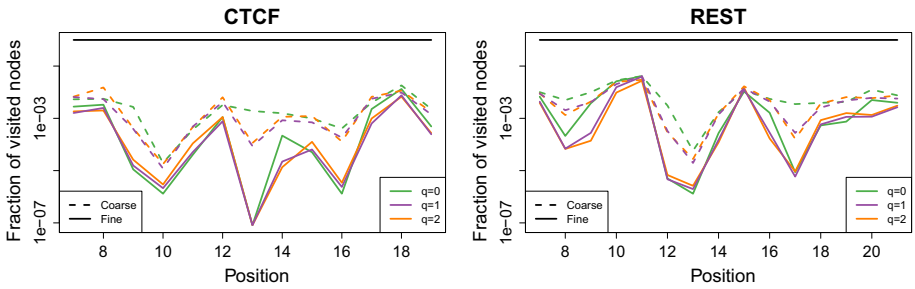

Fig. 9 Comparison of different pruning variants. Shown is the reduction of the search space in relation to basic DP in terms of the number of visited nodes of the extended PCT. Here we compare the coarse with the fine upper bound and different values for q, the number of lookahead steps (Color figure online)

We observe that the biggest difference among methods is achieved at seemingly "easy" positions: the most striking example is again CTCF-13, where the difference among the best and the worse pruning technique amounts to four orders of magnitude. Moreover, the switch between the coarse and fine upper bound has a higher impact than changing the number of lookahead steps. Except for a few difficult cases (CTCF-18, REST-11, REST-15) using the fine bound has always a clearly positive effect on the reduction of the search space, and it never increases the work load in terms of the number of visited nodes.

Lookahead, however, can have a negative effect, as it potentially increases the search space in cases where it has little benefit on tightening the bounds. With the coarse upper bound, lookahead clearly pays off, $q = 1$ and $q = 2$ are both almost equally good and in some cases (CTCF-13, REST-17) substantially better than $q = 0$. With the fine upper bound, $q = 1$ performs best. For a few positions (CTCF-14, REST-9, REST-16), the one-step lookahead substantially improves the shallow fine upper bound by more than one order of magnitude. Furthermore, for the majority of positions $q = 1$ is slightly superior to $q = 2$, but there are a few instances where further lookahead pays off, such as REST-9 or REST-10. The cases $q > 2$, we omit from the plots for clarity, follow the trend from $q = 1$ to $q = 2$ and yield inferior performance.

We conclude that the fine upper bound in combination with one-step lookahead is a competitive choice. Two-step lookahead is for these data sets not substantially worse, as the additional number of visited lookahead nodes is compensated by the tighter bound so the parameter is robust.

7.4 The AIC score

In the previous two sections, we used the BIC score as the objective function to be optimized. It is a reasonable choice in the domain of DNA binding site modeling due to its harsh penalty term (Eggeling et al. 2014b), which yields sparse trees as shown in Fig. 7. Now, we repeat the study from Sect. 7.2, but replace BIC by AIC, which is known penalize complex models less heavily. While we refrain from showing the optimal PCTs for brevity, they are indeed substantially more complex in terms of the number of leaves: for CTCF the mean over all sequence positions is 11.4 and the median is 8, for REST the mean is 12.1 and the median is 13.

We again compute optimal PCTs of depth $d = 6$ for all algorithm variants. The results are shown in Fig. 10. The savings for memoization are exactly the same as in the case of the

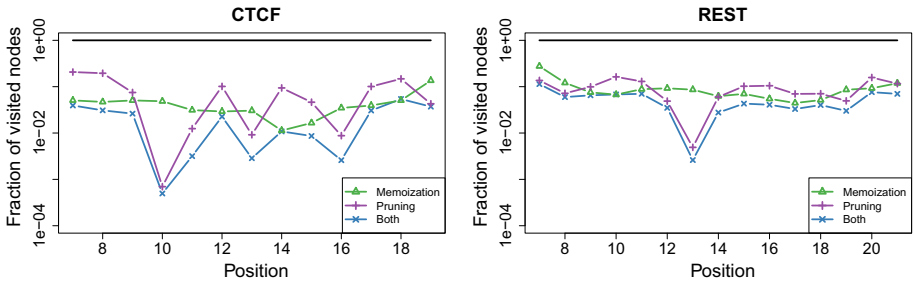

Fig. 10 Savings with the AIC score. The plot shows the fraction of visited nodes of extended PCT for different algorithm variants in analogy to Fig. 8, from whom it differs only in the scoring function and the range of the y-axis (Color figure online)

BIC score, which serves as a sanity check: the memoization technique does not distinguish between BIC and AIC, and so the results must be identical.

The results for pruning, however, dramatically change. Due to the less harsh penalty term, total statistical independence never occurs, that is, the minimal tree is never optimal. Moreover, context-specific independence can be declared in much fewer cases than for BIC, and so the pruning rules are less effective. The largest saving occurs for CTCF-10, where the AIC-optimal PCT has only four leaves, the saving being a little more than three orders of magnitude, which is comparable to the worst cases for BIC on the same data set. There are even instances, where the reduction of the search space is smaller than one order of magnitude.

The comparatively poor effect of the pruning rules, however, changes the game when pruning is combined with memoization. While is some cases like CTCF-10, Rest-8, or REST-8 pruning alone could still suffice, and in a few other cases like CTCF-14, CTCF-18, or REST-10 memoization alone yields already the best possible result, combining the two ideas clearly pays off for the majority of positions. It demonstrates that the memoization idea can in principle be as valuable as pruning or be even more effective, depending heavily on the scoring function and the complexity of the optimal model structures.

7.5 Memoization revisited

As demonstrated in the last section, the memoization technique has the merit of yielding a certain reduction of the search space, no matter whether the scoring function favors for sparse or complex models. However, memoization has the downside that storing solutions to previously computed subproblems—either scores associates with data subsets or even entire subtrees—can substantially increase the memory consumption.

We thus investigate the impact of the memoization depth m, which indicates the deepest layer of the extended PCT for which subproblems are stored for potential re-use later on. For measuring time consumption, we count the number of *visited* nodes in the extended PCT. Since the total running time for a data set is the main factor of interest, we here take the *mean* value over all positions. For measuring space consumption, we count the number of *stored* nodes. Here, however, we take the *maximum* over all positions, since it typically is the quantity of interest to decide whether a problem can be solved on a given machine or not. Figure 11 displays the results.

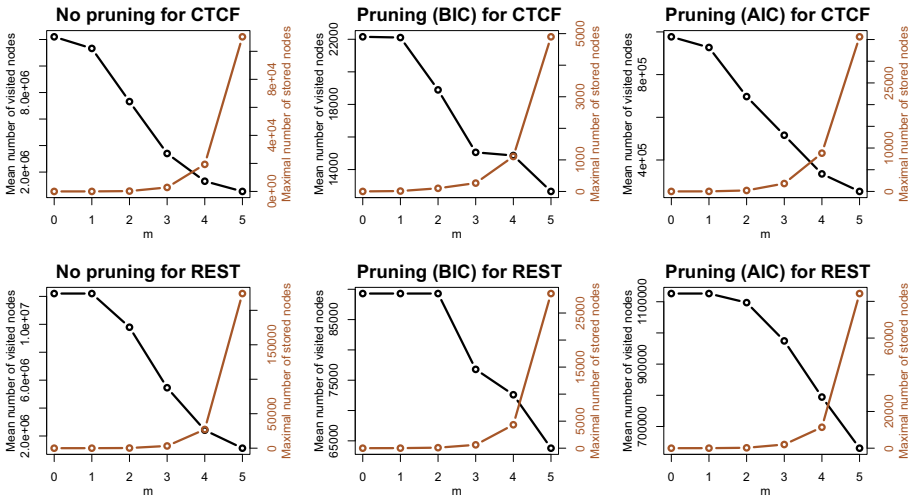

Fig. 11 Effect of memoization depth for finding optimal PCTs of maximum depth $d = 6$. Memoization depth $m = 0$ disables memoization entirely, whereas $m = 5$ enables maximal memoization (Color figure online)

We observe that the pattern is similar for all six cases, and $m = 4$ gives the overall best tradeoff between time and space complexity. For cases where pruning is rather effective, such as BIC, space complexity may not become a critical bottleneck, so even $m = 5$ could be justified. In the other cases, it might be a good idea to stop storing subproblems one layer earlier by setting $m = d - 2$ and to compute, if needed, the optimal partition of the leaf nodes of the extended PCT explicitly.

7.6 Broad study

In the previous sections, we investigated two data sets in detail and used the number of visited nodes in the extended PCT as an evaluation metric. Two open questions remain: How do the numbers of visited nodes translate to running times? How does the algorithmic variants perform on a larger variety of data sets, in particular with respect to the sample size?

In order to shed light on these issues, we now investigate 95 data sets with varying sample size, from $N = 102$ to $N = 8{,}734$ (see Appendix for the full list). We use the BIC score as objective function, the fine upper bound with the lookahead parameter $q = 1$ as pruning method, and full memoization. The sequence length, which determines the number of PCTs to be optimized, differs among data sets. Using $d = 6$, we learn all 767 PCTs and plot the running times required for each of the four algorithmic variants in Fig. 12(left). Performing a signed-rank test of Wilcoxon (1945) among these variants, we find that all pairwise differences are highly significant with p values below 10^{-10}.

The results generally confirm the observation from the previous sections: pruning gives larger savings than memoization, even though the difference in running times is not as large as the difference in the number of visited nodes (Sect. 7.2). One explanation is that the computation of the fine upper bound does have a certain computational cost, whereas memoization has a memory- rather than a computation-overhead. In addition, memoization can also give improvements in cases where pruning itself is ineffective. As a consequence, the

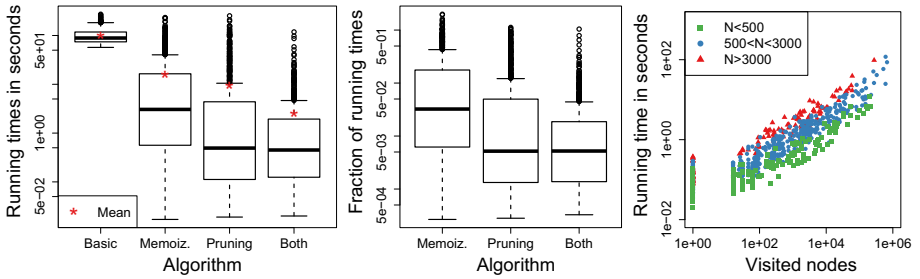

Fig. 12 Broad study of finding optimal PCTs of maximum depth $d = 6$ for 767 problem instances originating from 95 sequence data sets. Left: boxplot over the running times for different algorithmic variants. Middle: boxplot over the savings in running times in relation to the basic algorithm. Right: the relationship between running times and numbers of visited nodes of the extended PCT for the combination of pruning and memoization; the color indicates the sample size of the underlying data set (Color figure online)

combination of pruning and memoization is the significantly best choice for speeding up PCT optimization and reducing the median running time by almost two orders of magnitude.

In Fig. 12(right), we plot for this best variant the running time against the number of visited nodes in the extended PCT, for each of the 767 problem instances. We color each point in the scatter plot by the size of the data set, distinguishing three size groups, roughly on a log scale: small with $N < 500$, typical with $500 < N < 3000$, and large with $N > 3000$, consisting of 23, 52, and 20 data sets respectively (each amounting to several instances). We observe that the running time correlates well with the number of visited nodes (Pearson correlation coefficient $\rho = 0.90$).

One factor that prohibits a perfect linear correlation between running times and visited nodes is the sample size N, which itself has a roughly linear effect on the running time. This is because all data points need to be read and distributed among the nodes in the extended PCT, which becomes most evident for all cases where pruning applies directly at the root (1 visited extended PCT node) where the correlation between sample size and running time is almost perfect ($\rho = 0.99$). For the remaining cases, the relationship is less perfect but the general trend remains the same. For the four-symbol alphabet the data management, as opposed to the alphabet partitioning, dominates the workload in each node of the extended PCT.

7.7 Running times for different parameter values

The previous section discussed the running times for concrete selections the algorithms' parameters. We now set these parameters, one at a time, to possible alternative values and study the effects on the running time (Fig. 13). We observe that for every parameter there exist some problem instances that benefit from a change of the parameter value, but nevertheless we do observe a general trend.

When using the coarse bound instead of the fine bound (top, right), we find that for the majority of problem instances the running time increases, and in many cases by more than one order of magnitude. Keeping the fine bound, but disabling the lookahead instead (top, center) also leads to an increased running time for the majority of instances. These are often cases the minimal PCT is optimal (red), and whereas the fine bound enables pruning directly very early in the optimization, the coarse bound does not. Increasing the lookahead from

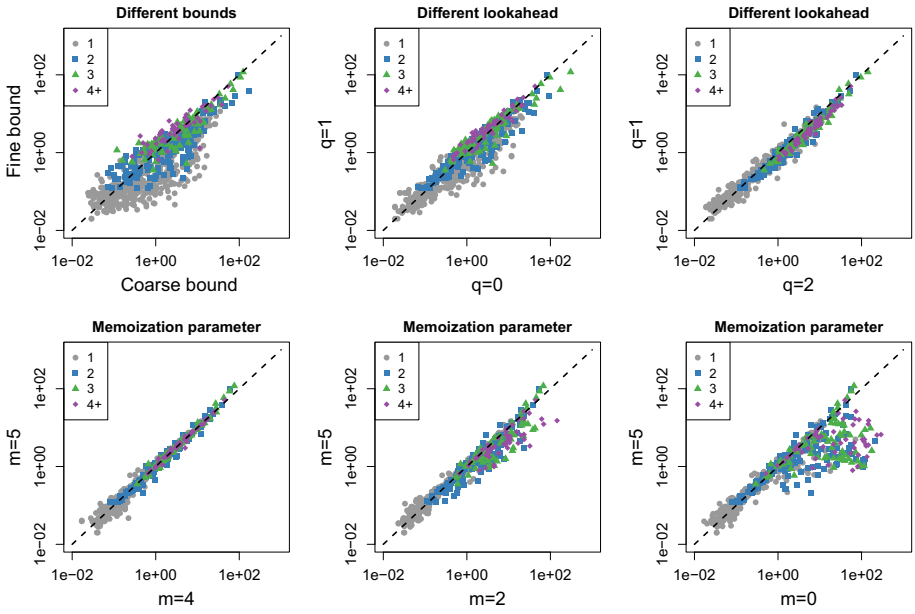

Fig. 13 Effect of different parameter settings on the running time in seconds. Y-axis always corresponds to the full algorithm (fine bound, one-step lookahead, full memoization). Each plot varies exactly one of these three parameters to a different value. Legend indicates number leaves in optimal PCT (Color figure online)

$q = 1$ to $q = 2$ has relatively little effect, and thus confirms the expectations gained from analyzing the number of visited nodes (cf. Fig. 9).

When varying the memoization parameter m, we observe that for the majority of problem instances the running times remain widely identical, especially these where the optimal PCT has only one leaf. However, for many instances where the optimal PCT has more than one leaf, gradually disabling memoization by reducing m increases running time. These results also confirm the expectations from the earlier analysis: pruning and memoization complement each other: whereas the former technique attempts to quickly identify context-specific independencies (including complete independence), the latter allows savings also in cases where the optimal PCT is relatively complex.

7.8 Predictive performance

Armed with the algorithmic tricks described in this paper, we are now able to study the predictive performance of a PCT-based model, dubbed iPMM (inhomogeneous parsimonious Markov model), on a large scale. We also investigate the performance of Bayesian networks (BNs), which have been previously proposed for the modeling complexity in transcription factor binding sites (Barash et al. 2003). This comparison is particularly relevant as the two model classes take into account different features in the data: iPMMs allow dependencies only among nucleotides in close proximity, but they model such dependencies in a very sparse and efficient way. BNs also allow long-range dependencies among distant positions in the sequence, but they are potentially less effective for short-range dependencies due to their use of conditional probability tables.

Table 2 Number of instances for which an iPMM predicts better/worse than a BN

	$\alpha = 0.05$	$\alpha = 0.005$	$\alpha = 0.0005$
Better	75	70	66
Tie	11	16	22
Worse	9	9	7

To allow for a fair comparison among the structural features of both model classes, we learn globally optimal iPMMs and BNs with the same structure score (BIC) and the same parameter estimator given the structure (posterior mean with pseudo count 1/2). For BN structure learning, we use an implementation of the dynamic programming algorithm of Silander and Myllymäki (2006), which is sufficient for finding a globally optimal DAG for the problem sizes within this application domain. For evaluating the predictive performance for both models we employ a repeated holdout approach with 90% training data and 100 repetitions. For each data set, we compute the mean log predictive probabilities and test whether the difference among both models is significant using the signed-rank test of Wilcoxon (1945). The individual results for all 95 data sets under consideration are shown in the Appendix.

Table 2 summarizes these results for a few different significance levels. We find that iPMMs describe the majority of data sets significantly better than BNs, which justifies the use of a PCT-based model. Data sets with long-range dependencies among nucleotides, which cannot be taken into account by iPMMs, exist, but they are the exception rather than the rule.

While the absolute difference among the predictive probabilities may seem small, the practical relevance depends on the concrete application. For scanning an entire genome with a threshold-based approach, for instance, even small differences in the predictive probability may have a substantial impact on the number of false positives. In addition to the general advantages such as easy visualization as discussed in Sect. 1, iPMMs have also the conceptual advantage over BNs that the running time grows only linear with the sequence length. Hence, they could be used to model longer sequence patterns, while still retaining optimality with respect to the chosen objective function.

7.9 Other types of data

Since DNA binding site data (1) concerns only $|\Omega| = 4$ and (2) entails some highly-informative response variables due to the inhomogeneity of the used iPMM, they may be not fully representative for other types of data. We thus additionally investigate our algorithmic techniques on learning PCTs from protein sequences, which are typically described using the 20-letter amino acid alphabet. However, for many applications it is common to reduce this alphabet to smaller sizes based on, e.g., similar biochemical properties of certain amino acids (Li et al. 2003; Peterson et al. 2009; Bacardit et al. 2009). In this study we use the alphabet reduction method of Li et al. (2003), since it offers for each possible reduced alphabet size an optimal clustering of amino acids into groups.

We study protein sequences from the UniProt database (The UniProt Consortium 2017). In order to somewhat limit the number of data sets, we consider only human proteins with catalytic activity. In addition, we data sets to these with a protein length between 250 and 500 residues, which is motivated by the median human protein length of 375 (Brocchieri and Karlin 2005). We further exclude three selenoproteins and finally retain 1191 sequences.

For each of these sequences, we learn a PCT (thus implicitly assuming a homogeneous model) with the basic DP algorithm and with our full algorithm with improvements enabled

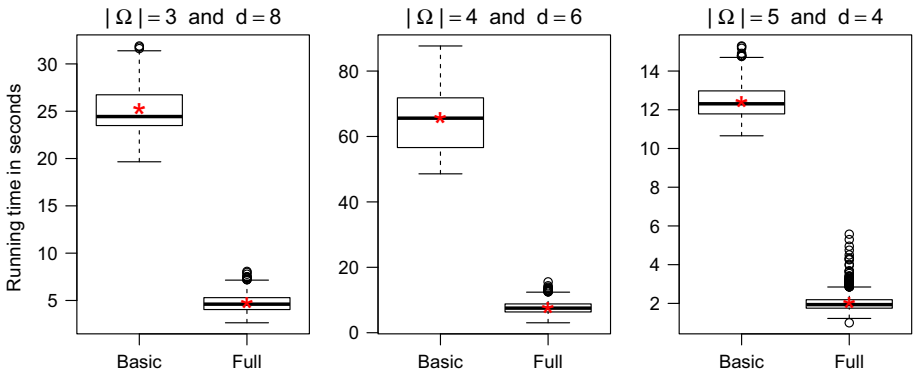

Fig. 14 Running time comparison on protein sequences from UniProt (Color figure online)

Table 3 Algorithm comparison for learning PCTs of depth d = 4 on ADL data

Data set	*N*	Metric	Basic	Full	Saving factor
A	248	VN	262,209,281	15,007,453	17.47
A	248	RT	6451	598	10.78
B	493	VN	262,209,281	24,514,291	10.69
B	493	RT	6955	1129	6.16

The metrics are the running time in seconds (RT) and the number of visited nodes in the extended PCT (VN)

and plot the required running time for three combinations of alphabet size and maximal PCT depth in Fig. 14. We find that our algorithm speeds up structure learning also for this type of data and model. Compared to the results on DNA binding sites, the savings rates are on smaller on average, but also the variance in savings is decreased. This can be explained by the observation that in homogeneous sequences the response variables rarely have an extreme marginal distribution, so pruning at or close to the root almost never occurs, even if the independence models was optimal.

In order to also consider an example from a non-biological domain, we evaluate the PCT learning algorithms on the *activity of daily living (ADL)* data of Ordonéz et al. (2013), obtained from the UCI machine learning repository (Lichman 2013). We extract the sequence of daily activities of both users, which contain nine and ten possible states, respectively. We further combine the states "Breakfast", "Lunch", "Snack", and "Dinner" (the latter occurs only for user B) into a single state "Meal", thus obtaining two sequences with alphabet size seven. We use these data for learning PCTs of depth four for both ADL data sets with our algorithm and with the basic variant and display the results in Table 3. We again obtain a substantial reduction of search space and running time that is comparable to the previous results on protein sequences.

8 Discussion

We have investigated the problem of learning parsimonious context trees (Bourguignon and Robelin 2004), which are a powerful model class for sequential discrete data, but entail the challenge of requiring high computational effort for exact structure learning (Leonardi 2006). Specifically, we proposed two orthogonal ideas to expedite the basic dynamic programming

algorithm of Bourguignon and Robelin (2004) for finding a highest-scoring parsimonious context tree.

The first idea, memoization, exploits regularities among the explanatory variables by storing and re-using previously optimized subtrees. Empirical results on real world DNA binding site data suggest that memoization reduces the search space by about one order of magnitude in typical cases. The variance in the savings factor is generally moderate since regularities among multiple explanatory variables need to coincide for the memoization rule to apply; extreme cases with extraordinary high regularity are rare unless the dependence among variables is actually deterministic. While memoization is rather memory-intensive in its maximal instantiation, we have seen that the memory burden can be significantly reduced by putting a limit on the number of stored subproblem solutions, losing only little in search space reduction. We observed that a simple implementation of this idea works: storing solutions only up to a certain user-specified depth provides an interpolation between the minimal and maximal memory requirements. Let us note that we also investigated several alternative criteria for deciding whether the solution to a particular subproblem should be stored for later re-use or not, such as the number of (distinct) data points matching the corresponding node. However, no other criterion could compete with the simple depth-based criterion.

The second idea, pruning, exploits regularities within the response variable through upper bounds on the scoring function. Specifically, we derived local score upper bounds for scores with a constant leaf penalty such as the BIC score or the AIC score, with an option for a few-step lookahead. We presented two pruning rules that utilize these upper bounds: a stopping rule and a deletion rule. Empirical results showed that pruning can be extremely effective when the entropy in the response variable is low and the scoring function favors sparse trees, which is typically the case for BIC. Here, pruning substantially outperforms memoization and when employing both, the latter appears to offer only a negligible additional contribution. However, the reduction of the traversed search space is less pronounced when the distribution of the response variable has a high entropy and when the found optimal tree is large. In this situation, the combination of pruning and memoization pays off.

The effectiveness of pruning for a given scoring function depends partially on the quality of the score upper bounds. We may control and influence this aspect by *algorithmic decisions* concerning the amount of effort we are willing to invest for getting tighter bounds. However, our case studies demonstrate that it additionally depends on the size of the tree structures favored by the scoring function: if the optimal tree is sparse, then there is more potential, albeit no guarantee, for pruning large parts of the search space. This second aspect is beyond our direct control in algorithm design once data set and scoring function are fixed. As a consequence, the choice of the scoring function, a pure *modeling decision*, has a direct and in fact rather predictable impact on the speed of the algorithm.

It might be noteworthy that using the BIC score for learning parsimonious context trees was previously motivated from the perspective of predictive performance under limited data (Eggeling et al. 2014b). The empirical results from this work now also suggest an algorithmic justification for this scoring function choice, and it can be considered as a fortunate coincidence that these two different objectives lead to the same conclusion.

While our case studies involved only two concrete scoring functions and one type of benchmark data, we believe that the lessons learned can be somewhat generalized. Since our score upper bounds for BIC and AIC share the same functional form, we may assume that the upper bounds alone are roughly equally effective in both cases. Hence, a larger leaf penalty of a scoring function implies a larger pruning potential on average. This conclusion should generalize also to other scoring functions, such as Bayes scores arising from different prior

choices, albeit deriving effective upper bounds could be technically more challenging in these cases. In contrast, memoization makes only weak assumptions about the scoring function and is thus always equally effective no matter whether model complexity is penalized heavily or not. This interplay of pruning and memoization techniques is likely to generalize also to other classes tree-structured probabilistic models.

One obvious limitation of the method is that the effectiveness of the proposed methods wane with growing sample size as memoization becomes less likely to apply and also the optimal PCTs become larger. However, we find this limitation not very significant, as the very purpose of PCTs is to provide a sparse representation of a conditional distribution in small data scenarios where a full Markov model would have excessively many parameters. Thus, it may be not critical if learning the model becomes computationally infeasible in situations where the model does not have clear advantage over computationally simpler models in the first place.

Another limitation is that the presented methods, as such, are insufficient for handling large alphabets. The reason is that increasing the alphabet size does not only increase the size of the extended PCT (which could be dealt with by pruning and memoization), but also the time needed for computing a single optimal partition for each of its inner nodes. Since the time complexity for the latter task is already $\mathcal{O}(3^{|\Omega|})$, it does not seem likely that exact learning of PCTs on more than a dozen of symbols becomes tractable for practically relevant instances. So if handling a large alphabet is critical for the specific application and cannot be circumvented by some alphabet reduction method, one should resort either to heuristic algorithms or to simpler and potentially less powerful models.

The present work also opens avenues for future research. For instance, it might be worthwhile to apply the pruning ideas for finding optimal classification and regression trees (Breiman et al. 1984; Buntine 1992; Chipman et al. 1998) with many categorical explanatory variables. The published exact algorithms for learning decision trees (Blanchard et al. 2007; Hush and Porter 2010; Bertsimas and Dunn 2017) do not employ pruning based on score upper bounds; the pruning strategies explored in the literature—see, e.g., Frank (2000), Lomax and Vadera (2013), and references therein—are limited to post-processing of decision trees found by greedy, inexact algorithms, and are thus not comparable to the methods presented in the present paper. It also remains to be investigated whether the bound-and-prune approach could succeed in expediting other algorithms that are based on recursive set partitioning. An example is the DP algorithm by Kangas et al. (2014) for learning chordal Markov networks, for which Rantanen et al. (2017) recently presented a related bound-and-prune variant; however, the variant appears to not take full advantage of the underlying DP algorithm and yields speedups only occasionally. A different line of research is to design heuristic, approximate algorithms for learning parsimonious context trees that scale to large alphabets and thereby significantly widen the applicability of the model class. We believe the techniques presented and the insight obtained in this work constitute a fruitful starting point for devising effective greedy and local search algorithms.

Acknowledgements Open access funding provided by University of Helsinki including Helsinki University Central Hospital.

Funding Funding was provided by Academy of Finland (Grant No. 276864).

Appendix

The following table shows the negative mean log predictive probabilities obtained by Bayesian network (BN) and inhomogeneous parsimonious Markov model (iPMM) for 95 data sets of transcription factor binding sites. The values are normalized by number of data points N and sequence length L. Random guessing would thus achieve a value of $-\ln(0.25) \simeq 1.3863$, and smaller values indicate thus improved predictive performance.

ID	Name	L	N	BN	iPMM	p value
MA0079.3	SP1	11	8734	0.5379 ± 0.0005	0.5336 ± 0.0004	3.96E−18
MA0479.1	FOXH1	11	8211	0.4989 ± 0.0003	0.4946 ± 0.0003	3.96E−18
MA0095.2	YY1	12	7171	0.5354 ± 0.0005	0.5360 ± 0.0005	4.93E−06
MA0512.1	Rxra	11	5348	0.5599 ± 0.0004	0.5439 ± 0.0004	3.96E−18
MA0478.1	FOSL2	11	5318	0.4903 ± 0.0005	0.4879 ± 0.0004	2.97E−16
MA0477.1	FOSL1	11	5272	0.4726 ± 0.0004	0.4816 ± 0.0004	3.96E−18
MA0105.3	NFKB1	11	5112	0.5206 ± 0.0005	0.5152 ± 0.0004	3.96E−18
MA0003.2	TFAP2A	15	5098	0.7731 ± 0.0005	0.7719 ± 0.0005	3.67E−09
MA0506.1	NRF1	11	4624	0.4800 ± 0.0008	0.4744 ± 0.0008	3.96E−18
MA0508.1	PRDM1	15	4603	0.6713 ± 0.0007	0.6685 ± 0.0008	4.89E−18
MA0036.2	GATA2	14	4380	0.6986 ± 0.0005	0.6956 ± 0.0005	7.44E−18
MA0523.1	TCF7L2	14	4188	0.7052 ± 0.0005	0.7015 ± 0.0005	4.08E−18
MA0529.1	BEAF-32	15	3985	0.7528 ± 0.0004	0.7503 ± 0.0004	1.62E−17
MA0510.1	RFX5	15	3868	0.7389 ± 0.0006	0.7344 ± 0.0006	4.08E−18
MA0475.1	FLI1	11	3667	0.5407 ± 0.0006	0.5364 ± 0.0006	1.39E−17
MA0137.3	STAT1	11	3629	0.4478 ± 0.0008	0.4473 ± 0.0007	2.23E−02
MA0141.1	Esrrb	12	3605	0.6216 ± 0.0009	0.6206 ± 0.0008	2.15E−13
MA0503.1	Nkx2-5(var.2)	11	3429	0.5780 ± 0.0005	0.5751 ± 0.0005	1.82E−17
MA0076.2	ELK4	11	3427	0.5515 ± 0.0005	0.5510 ± 0.0006	1.39E−02
MA0537.1	blmp-1	11	3368	0.5129 ± 0.0007	0.4991 ± 0.0006	3.96E−18
MA0518.1	Stat4	14	2873	0.7010 ± 0.0006	0.6969 ± 0.0006	3.96E−18
MA0471.1	E2F6	11	2757	0.5426 ± 0.0006	0.5382 ± 0.0006	4.89E−18
MA0482.1	Gata4	11	2746	0.5337 ± 0.0005	0.5306 ± 0.0005	4.74E−18
MA0469.1	E2F3	11	2549	0.5066 ± 0.0007	0.5046 ± 0.0007	1.10E−10
MA0480.1	Foxo1	11	2490	0.5371 ± 0.0007	0.5364 ± 0.0007	4.44E−03
MA0216.2	cad	11	2303	0.5283 ± 0.0006	0.5190 ± 0.0006	3.96E−18
MA0507.1	POU2F2	13	2287	0.5645 ± 0.0008	0.5625 ± 0.0008	7.89E−12
MA0083.2	SRF	18	2277	0.7744 ± 0.0008	0.7683 ± 0.0009	3.96E−18
MA0497.1	MEF2C	15	2209	0.7323 ± 0.0007	0.7280 ± 0.0007	4.08E−18
MA0541.1	efl-1	15	2206	0.7445 ± 0.0010	0.7338 ± 0.0011	4.33E−18
MA0509.1	Rfx1	14	2138	0.6244 ± 0.0009	0.6128 ± 0.0009	3.96E−18
MA0467.1	Crx	11	2097	0.5642 ± 0.0007	0.5584 ± 0.0007	5.35E−18
MA0137.2	STAT1	15	2069	0.7729 ± 0.0010	0.7701 ± 0.0010	3.46E−14
MA0531.1	CTCF	15	1902	0.7302 ± 0.0012	0.7252 ± 0.0012	6.22E−18
MA0470.1	E2F4	11	1878	0.5526 ± 0.0008	0.5454 ± 0.0008	3.96E−18
MA0098.2	Ets1	15	1868	0.7314 ± 0.0008	0.7321 ± 0.0008	1.52E−04
MA0520.1	Stat6	15	1852	0.7232 ± 0.0007	0.7220 ± 0.0006	1.03E−07
MA0483.1	Gfi1b	11	1761	0.5271 ± 0.0007	0.5263 ± 0.0007	3.35E−02
MA0505.1	Nr5a2	15	1702	0.6775 ± 0.0013	0.6719 ± 0.0013	5.73E−17
MA0516.1	SP2	15	1686	0.7535 ± 0.0010	0.7488 ± 0.0010	6.03E−18
MA0940.1	Foxo1	13	1622	0.7565 ± 0.0010	0.7516 ± 0.0010	5.04E−18
MA0465.1	CDX2	11	1597	0.5033 ± 0.0007	0.5055 ± 0.0007	1.92E−12
MA0138.2	REST	21	1575	0.6036 ± 0.0014	0.5992 ± 0.0015	2.43E−16

ID	Name	L	N	BN	iPMM	p value
MA0052.2	MEF2A	15	1473	0.6863 ± 0.0009	0.6842 ± 0.0009	1.35E−13
MA0050.2	IRF1	21	1362	0.7768 ± 0.0013	0.7765 ± 0.0013	3.00E−01
MA0142.1	Pou5f1::Sox2	15	1356	0.6975 ± 0.0013	0.6926 ± 0.0014	1.48E−17
MA0452.2	Kr	14	1296	0.6791 ± 0.0009	0.6798 ± 0.0010	1.04E−03
MA0494.1	Nr1h3::Rxra	19	1269	0.8804 ± 0.0010	0.8792 ± 0.0010	7.85E−07
MA0543.1	eor-1	15	1253	0.6947 ± 0.0015	0.6918 ± 0.0016	8.99E−11
MA0472.1	EGR2	15	1246	0.7227 ± 0.0012	0.7179 ± 0.0013	2.80E−15
MA0106.2	TP53	15	1231	0.5836 ± 0.0015	0.5768 ± 0.0016	6.44E−17
MA0501.1	MAF::NFE2	15	1090	0.5564 ± 0.0014	0.5553 ± 0.0014	7.39E−04
MA0511.1	RUNX2	15	1062	0.7556 ± 0.0012	0.7532 ± 0.0012	7.06E−07
MA0024.2	E2F1	11	1059	0.5571 ± 0.0009	0.5533 ± 0.0010	1.63E−16
MA0062.2	Gabpa	11	987	0.5441 ± 0.0018	0.5388 ± 0.0019	1.16E−13
MA0463.1	Bcl6	14	956	0.7078 ± 0.0011	0.7079 ± 0.0012	9.08E−01
MA0139.1	CTCF	19	908	0.7231 ± 0.0014	0.7171 ± 0.0014	4.46E−18
MA0513.1	SMAD2::SMAD3	13	899	0.6493 ± 0.0013	0.6474 ± 0.0013	2.13E−09
MA0014.2	PAX5	19	896	0.8474 ± 0.0017	0.8383 ± 0.0018	4.04E−17
MA0554.1	SOC1	15	888	0.6939 ± 0.0013	0.6930 ± 0.0013	2.01E−03
MA0148.1	FOXA1	11	888	0.5938 ± 0.0017	0.5911 ± 0.0017	4.71E−10
MA0485.1	Hoxc9	13	885	0.6575 ± 0.0010	0.6548 ± 0.0010	3.46E−10
MA0536.1	pnr	11	869	0.5012 ± 0.0012	0.4983 ± 0.0013	9.59E−13
MA0065.2	Pparg::Rxra	15	855	0.8386 ± 0.0015	0.8363 ± 0.0015	2.20E−08
MA0144.1	Stat3	19	821	0.8324 ± 0.0014	0.8299 ± 0.0014	3.31E−10
MA0047.2	Foxa2	12	800	0.6212 ± 0.0016	0.6125 ± 0.0016	1.52E−17
MA0150.2	Nfe2l2	15	726	0.6305 ± 0.0017	0.6310 ± 0.0018	2.35E−01
MA0527.1	ZBTB33	15	705	0.7258 ± 0.0016	0.7242 ± 0.0016	6.14E−05
MA0143.1	Sox2	15	662	0.7755 ± 0.0019	0.7748 ± 0.0019	4.26E−01
MA0517.1	STAT1::STAT2	15	620	0.6230 ± 0.0019	0.6181 ± 0.0019	4.99E−13
MA0559.1	PI	14	558	0.6916 ± 0.0021	0.6915 ± 0.0021	1.59E−01
MA0493.1	Klf1	11	526	0.5263 ± 0.0019	0.5252 ± 0.0020	3.35E−02
MA0530.1	cnc::maf-S	15	474	0.7456 ± 0.0014	0.7408 ± 0.0015	8.92E−14
MA0146.1	Zfx	20	468	0.8954 ± 0.0016	0.8962 ± 0.0017	5.60E−02
MA0112.2	ESR1	20	467	0.9214 ± 0.0018	0.9158 ± 0.0018	1.16E−14
MA0504.1	NR2C2	15	395	0.6469 ± 0.0022	0.6440 ± 0.0023	1.18E−07
MA0545.1	hlh-1	11	381	0.5632 ± 0.0019	0.5647 ± 0.0020	2.58E−03
MA0258.1	ESR2	18	357	0.8691 ± 0.0022	0.8639 ± 0.0023	1.24E−10
MA0547.1	skn-1	15	318	0.6721 ± 0.0030	0.6643 ± 0.0032	1.83E−12
MA0481.1	FOXP1	15	311	0.7310 ± 0.0016	0.7161 ± 0.0020	4.51E−14
MA0544.1	snpc-4	12	310	0.4672 ± 0.0038	0.4595 ± 0.0037	6.21E−16
MA0556.1	AP3	15	291	0.7305 ± 0.0023	0.7307 ± 0.0023	4.29E−01
MA0558.1	FLC	21	275	0.8672 ± 0.0019	0.8563 ± 0.0020	2.51E−15
MA0486.1	HSF1	15	225	0.6884 ± 0.0028	0.6894 ± 0.0029	8.72E−02
MA0538.1	daf-12	15	156	0.7136 ± 0.0042	0.6573 ± 0.0053	6.60E−18
MA0550.1	BZR1	14	153	0.7472 ± 0.0024	0.7498 ± 0.0025	1.21E−04
MA0548.1	AGL15	15	150	0.7638 ± 0.0041	0.7636 ± 0.0040	1.09E−01
MA0563.1	SEP3	11	150	0.6932 ± 0.0043	0.6932 ± 0.0043	N/A
MA1012.1	AGL27	14	142	0.5404 ± 0.0027	0.5333 ± 0.0027	4.02E−11
MA0532.1	Stat92E	15	118	0.7456 ± 0.0032	0.7506 ± 0.0035	1.01E−04
MA0060.1	NFYA	16	116	0.8515 ± 0.0045	0.8457 ± 0.0047	7.67E−04
MA0552.1	PIF1	14	114	0.7638 ± 0.0025	0.7601 ± 0.0026	4.85E−07
MA0149.1	EWSR1-FLI1	18	105	0.1386 ± 0.0068	0.1379 ± 0.0070	2.44E−02
MA0534.1	EcR::usp	15	104	0.7380 ± 0.0041	0.7388 ± 0.0041	5.91E−02
MA0535.1	Mad	15	102	0.8290 ± 0.0025	0.8387 ± 0.0030	5.88E−11

References

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, *19*(6), 716–723.

Bacardit, J., Stout, M., Hirst, J., Valencia, A., Smith, R., & Krasnogor, N. (2009). Automated alphabet reduction for protein datasets. *BMC Bioinformatics*, *10*, 6.

Barash, Y., Elidan, G., Friedman, N., & Kaplan, T. (2003). Modeling dependencies in protein-DNA binding sites. In *Proceedings of the seventh annual international conference on research in computational molecular biology (RECOMB)* (pp 28–37).

Begleiter, R., El-Yaniv, R., & Yona, G. (2004). On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, *22*, 385–421.

Ben-Gal, I., Shani, A., Gohr, A., Grau, J., Arviv, S., Shmilovici, A., et al. (2005). Identification of transcription factor binding sites with variable-order Bayesian networks. *Bioinformatics*, *21*, 2657–2666.

Bertsimas, D., & Dunn, J. (2017). Optimal classification trees. *Machine Learning*, *106*(7), 1039–1082.

Blanchard, G., Schäfer, C., Rozenholc, Y., & Müller, K. (2007). Optimal dyadic decision trees. *Machine Learning*, *66*(2–3), 209–241.

Bourguignon, P. Y., & Robelin, D. (2004). Modèles de Markov parcimonieux: sélection de modele et estimation. In *Proceedings of the 5e édition des Journées Ouvertes en Biologie, Informatique et Mathématiques (JOBIM)*.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. In *Proceedings of the 12th conference on uncertainty in artificial intelligence (UAI)* (pp. 115–123).

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Belmont: Wadsworth.

Brocchieri, L., & Karlin, S. (2005). Protein length in eukaryotic and prokaryotic proteomes. *Nucleic Acids Research*, *33*(10), 3390–3400.

Bühlmann, P., & Wyner, A. (1999). Variable length Markov chains. *Annals of Statistics*, *27*, 480–513.

Buntine, W. (1992). Learning classification trees. *Statistics and Computing*, *2*(2), 63–73.

Chavira, M., & Darwiche, A. (2005). Compiling Bayesian networks with local structure. In *Proceedings of the 19th international joint conference on artificial intelligence (IJCAI)* (pp. 1306–1312).

Chickering, D., Heckerman, D., & Meek, C. (1997). A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of the 13th conference on uncertainty in artificial intelligence (UAI)* (pp. 80–89).

Chipman, H., George, E., & McCulloch, R. (1998). Bayesian CART model search. *Journal of the American Statistical Association*, *93*(443), 935–948.

de Campos, C., & Ji, Q. (2011). Efficient structure learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, *12*, 663–689.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, *39*(1), 1–38.

Eggeling, R. (2018). Disentangling transcription factor binding site complexity. *Nucleic Acids Research*. https://doi.org/10.1093/nar/gky683. **(epub ahead of print)**.

Eggeling, R., Gohr, A., Keilwagen, J., Mohr, M., Posch, S., Smith, A., et al. (2014a). On the value of intra-motif dependencies of human insulator protein CTCF. *PLoS ONE*, *9*(1), e85–629.

Eggeling, R., Grosse, I., & Grau, J. (2017). InMoDe: Tools for learning and visualizing intra-motif dependencies of DNA binding sites. *Bioinformatics*, *33*(4), 580–582.

Eggeling, R., & Koivisto, M. (2016). Pruning rules for learning parsimonious context trees. In *Proceedings of the 32nd conference on uncertainty in artificial intelligence (UAI)* (pp. 152–161).

Eggeling, R., Koivisto, M., & Grosse, I. (2015a). Dealing with small data: On the generalization of context trees. In *Proceedings of the 32nd international conference on machine learning (ICML)* (pp. 1245–1253).

Eggeling, R., Roos, T., Myllymäki, P., & Grosse, I. (2014b). Robust learning of inhomogeneous PMMs. In *Proceedings of the 17th international conference on artificial intelligence and statistics (AISTATS)* (pp. 229–237).

Eggeling, R., Roos, T., Myllymäki, P., & Grosse, I. (2015b). Inferring intra-motif dependencies of DNA binding sites from ChIP-seq data. *BMC Bioinformatics*, *16*, 375.

Frank, E. (2000). *Pruning decision trees and lists*. Ph.D. Thesis, University of Waikato, Department of Computer Science, Hamilton, New Zealand.

Fujimaki, R., & Morinaga, S. (2012). Factorized asymptotic Bayesian inference for mixture modeling. In *Proceedings of the 15th international conference on artificial intelligence and statistics (AISTATS)*.

Grau, J., Keilwagen, J., Gohr, A., Haldemann, B., Posch, S., & Grosse, I. (2012). Jstacs: A Java framework for statistical analysis and classification of biological sequences. *Journal of Machine Learning Research*, *13*, 1967–1971.

Heckerman, D., Geiger, D., & Chickering, D. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, *20*, 197–243.

Hush, D., & Porter, R. (2010). Algorithms for optimal dyadic decision trees. *Machine Learning*, *80*(1), 85–107.

Jaeger, M., Nielsen, J., & Silander, T. (2006). Learning probabilistic decision graphs. *International Journal of Approximate Reasoning*, *42*(1–2), 84–100.

Kangas, K., Koivisto, M., & Niinimäki, T. (2014). Learning chordal Markov networks by dynamic programming. In *Advances in neural information processing systems (NIPS)* (Vol. 27, pp. 2357–2365).

Leonardi, F. (2006). A generalization of the PST algorithm: Modeling the sparse nature of protein sequences. *Bioinformatics*, *22*(11), 1302–1307.

Li, T., Fan, K., Wang, J., & Wang, W. (2003). Reduction of protein sequence complexity by residue grouping. *Protein Engineering*, *16*, 323–330.

Lichman, M. (2013). *UCI machine learning repository*. Irvine, CA: University of California, School of Information and Computer Science. http://archive.ics.uci.edu/ml. Accessed 8 Oct 2018.

Lomax, S., & Vadera, S. (2013). A survey of cost-sensitive decision tree induction algorithms. *ACM Computing Surveys*, *45*(2), 16:1–16:35.

Nielsen, S. (2000). The stochastic EM algorithm: Estimation and asymptotic results. *Bernoulli*, *6*(3), 457–489.

Oliver, J. (1993). Decision graphs—an extension of decision trees. In *Proceedings of the 4th international workshop on artificial intelligence and statistics (AISTATS)* (pp. 343–350).

Ordonéz, F., de Toledo, P., & Sanchis, A. (2013). Activity recognition using hybrid generative/discriminative models on home environments using binary sensors. *Sensors*, *13*(5), 5460–5477.

Orenstein, Y., & Shamir, R. (2014). A comparative analysis of transcription factor binding models learned from PBM, HT-SELEX and ChIP data. *Nucleic Acids Research*, *42*(8), e63.

Peterson, E., Kondev, J., Theriot, J., & Phillips, R. (2009). Reduced amino acid alphabets exhibit an improved sensitivity and selectivity in fold assignment. *Bioinformatics*, *25*, 1356–1362.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Rantanen, K., Hyttinen, A., & Järvisalo, M. (2017). Learning chordal Markov networks via branch and bound. In *Advances in neural information processing systems* (NIPS), (Vol. 30, pp. 1845–1855).

Rissanen, J. (1983). A universal data compression system. *IEEE Transactions on Information Theory*, *29*(5), 656–664.

Sandelin, A., Alkema, W., Engström, P., Wasserman, W., & Lenhard, B. (2004). JASPAR: An open-access database for eukaryotic transcription factor binding profiles. *Nucleic Acids Research*, *32*, D91–D94.

Schneider, T., & Stephens, R. (1990). Sequence logos: A new way to display consensus sequences. *Nucleic Acids Research*, *18*(20), 6097–6100.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, *2*, 461–464.

Seifert, M., Gohr, A., Strickert, M., & Grosse, I. (2012). Parsimonious higher-order hidden Markov models for improved array-CGH analysis with applications to *Arabidopsis thaliana*. *PLOS Computational Biology*, *8*(1), e1002–286.

Shen, Y., Choi, A., & Darwiche, A. (2018). Conditional PSDDs: Modeling and learning with modular knowledge. In *Proceedings of the 33rd national conference on artificial intelligence (AAAI)* (pp. 6433–6442).

Silander, T., & Myllymäki, P. (2006). A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the 22nd annual conference on uncertainty in artificial intelligence (UAI)*.

Silander, T., Roos, T., & Myllymäki, P. (2010). Learning locally minimax optimal Bayesian networks. *International Journal of Approximate Reasoning*, *51*, 544–557.

Smith, J., & Anderson, P. (2008). Conditional independence and chain event graphs. *Artificial Intelligence*, *172*(1), 42–68.

Su, J., & Zhang, H. (2005). Representing conditional independence using decision trees. In *Proceedings of the 20th national conference on artificial intelligence (AAAI)* (pp. 874–879).

Teyssier, M., & Koller, D. (2005). Ordering-based search: A simple and effective algorithm for learning Bayesian networks. In *Proceedings of the 21st conference on uncertainty in artificial intelligence (UAI)* (pp. 584–590).

The UniProt Consortium. (2017). UniProt: The universal protein knowledgebase. *Nucleic Acids Research*, *45*(D1), D158–D169.

Tian, J. (2000). A branch-and-bound algorithm for MDL learning in Bayesian networks. In *Proceedings of the 16th conference on uncertainty in artificial intelligence (UAI)* (pp. 580–588).

Volf, P., & Willems, F. (1994). Context maximizing: Finding MDL decision trees. In *Proceedings of 15th symposium on information theory, Benelux* (pp. 192–200).

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, *1*(6), 80–83.

Zhao, X., Huang, H., & Speed, T. (2005). Finding short DNA motifs using permuted Markov models. *Journal of Computational Biology*, *12*, 894–906.